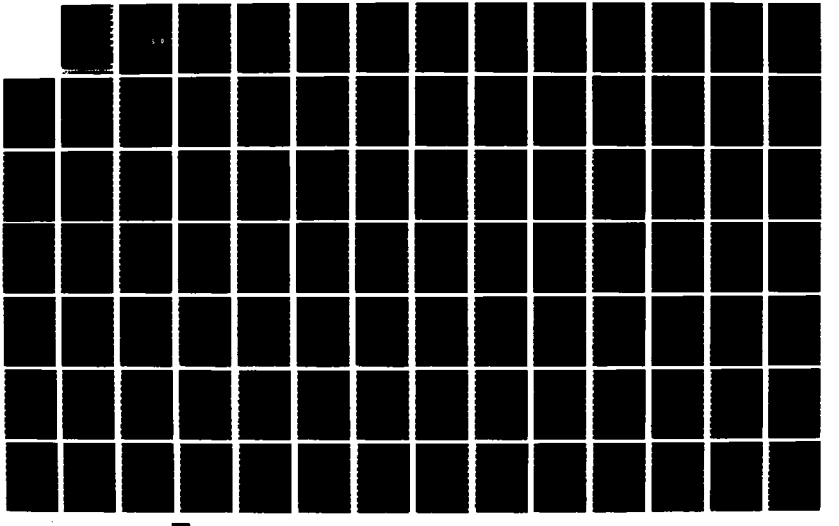
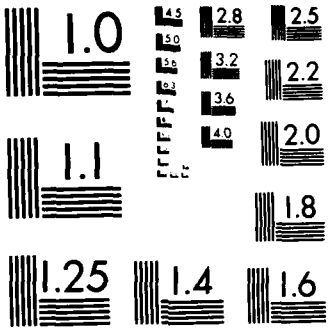
AD-A174 908 A DATABASE MANAGEMENT SYSTEM FOR ENGINEERING APPLICATIONS(U) IOWA UNIV IOWA CITY OPTIMAL DESIGN LAB Y SHYY ET AL 30 JUN 85 ODL-85 23 AFOSR-TR-86-2205 1/2

UNCLASSIFIED AFOSR-82-0322 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Technical Report No. ODL-85.23

A Database Management System For Engineering Applications

by

Y-K. Shyy, S. Mukhopadhyay and J. S. Arora

Optimal Design Laboratory
College of Engineering
The University of Iowa
Iowa City, IA 52242

Prepared for
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Air Force Systems Command, USAF
Under Grant No. AFOSR 82-0322

June 1985

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

ADA 174.908

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): ODL-85.23	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 86-2205
6a. NAME OF PERFORMING ORGANIZATION: Optimal Design Laboratory; 6b. OFFICE SYMBOL (If applicable): ODL	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NA
6c. ADDRESS (City, State and ZIP Code): College of Engineering, The University of Iowa, Iowa City, IA 52242	7b. ADDRESS (City, State and ZIP Code): Bolling AFB, D.C. 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Air Force Office of Scientific Research; 8b. OFFICE SYMBOL (If applicable): NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Grant No. AFOSR 82-0322
8c. ADDRESS (City, State and ZIP Code): Directorate of Aerospace Sciences, Bolling AFB, D.C. 20332-6448 (Bldg 410)	10. SOURCE OF FUNDING NOS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2307; TASK NO. B1; WORK UNIT NO.
11. TITLE (Include Security Classification): A Database Management System for Engineering Applications	
12. PERSONAL AUTHOR(S): Y-K. Shyy, S. Mukhopadhyay, and J.S. Arora	
13a. TYPE OF REPORT: Interim Technical; 13b. TIME COVERED: FROM 7-84 TO 6-85	14. DATE OF REPORT (Yr., Mo., Day): 1985-6-30; 15. PAGE COUNT: 162
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES: FIELD / GROUP / SUB GR	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Database Management, Design, Evaluation, Engineering,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes design of a database management system called MIDAS which stands for Management of Information for Design and Analysis of Systems. The system is designed to handle large matrix data of different types. A prototype system is first implemented. The system is re-designed to improve its efficiency. Capabilities of the system are described. It is evaluated in the engineering environment by solving systems of linear equations. The system is flexible; its buffer size, page size and number of pages can be changed. Variations of these parameters is studied and their effect on the system performance is evaluated. It is concluded that the application programs must be developed carefully to efficiently use a database management system. In addition, application programmer must be somewhat knowledgeable of the internal workings of the system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Anthony K. Amos	22b. TELEPHONE NUMBER (Include Area Code): 202-767-4937; 22c. OFFICE SYMBOL: NA

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

Technical Report No. ODL-85.23

A DATABASE MANAGEMENT SYSTEM FOR ENGINEERING APPLICATIONS

by

Y-K. Shyy, S. Mukhopadhyay and J.S. Arora

OPTIMAL DESIGN LABORATORY

College of Engineering
The University of Iowa
Iowa City, IA 52242

Prepared for

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Air Force Systems Command, USAF
Under Grant No. AFOSR 82-0322

June 1985

PREFACE

This report is the M.S. thesis of Mr. Y-K. Shyy completed under the supervision of Professor Jasbir S. Arora. Also Mr. Santanu Mukhopadhyay provided useful discussions and suggestions during the progress of this work. The research is sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF. The project title is "Database Design and Management in Engineering Design Optimization."

A DATABASE MANAGEMENT SYSTEM FOR ENGINEERING APPLICATIONS

by

Yaw-Kang Shyy

A thesis submitted on partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering
in the Graduate College of
The University of Iowa

May 1985

Thesis superviser: Professor Jasbir S. Arora

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Yaw-Kang Shyy

has been approved by the Examining Committee for the thesis requirement for the Master of Science degree in Mechanical Engineering at the May 1985 graduation.

Thesis committee: Jasbir S. Arora
Thesis supervisor

M. Asghar Bhatti
Member

Member

ACKNOWLEDGMENTS

In the first place, I wish to express my gratitude to Professor Jasbir S. Arora for his guidance and consideration; he showed me what a teacher really is. In addition I would like to thank Santanu Mukhopadhyay for his valuable criticism during the preparation of this manuscript. I would also thank my committee members, Professor M. Asghar Bhatti and Professor Kyung K. Choi for providing valuable advice on the thesis and for their careful reading of the manuscript. Their suggestions have greatly helped in improving this work.

The research reported in this thesis was supported by Air Force Office of Scientific Research Grant AFOSR-82-0322. Their support is gratefully acknowledged. I also wish to acknowledge the use of the excellent computing facilities at the Computer Aided Engineering Laboratory of The College of Engineering.

Finally, most of all, I wish to thank my mother and my wife for their constant encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

In recent years, number of papers have been published on the database management in scientific computing (Elliott, et al., 1978; Felippa, 1979, 1980; Messena, 1978; Rajan and Bhatti, 1983; SreekantaMurthy and Arora, 1983). The importance of using a database management system (DBMS) has been noted. Purpose of this thesis is to present a database management system for engineering applications. The system is called MIDAS/N which stands for Management of Information for Design and Analysis of Systems/Numerical Model. A preliminary design of the system called MIDAS/N Version 1.0 was implemented earlier. That design has been improved considerably and the new system is called MIDAS/N Version 2.0. The thesis presents these enhancements alongwith some applications.

Overview of MIDAS/N is given in Section 1.1. Purpose of using a database management system is given in Section 1.2. Purposes of developing MIDAS/N Version 1.0 and improving it to MIDAS/N Version 2.0 are described in Section 1.3. Section 1.4 describes some terminology about data structures used in MIDAS/N Version 2.0.

Improved capabilities, such as additional functions, error trapping, and system organization are discussed in Chapter 2. Purpose of isolating the application program (USER) interface in a separate

segment and its routines are given in Chapter 3. The application program interface processing system (APIPS) and its information table are described in Chapter 4. The memory management interface (MMI) is given in Chapter 5. In Chapter 6, storage allocation and replacement strategies used by the memory management system (MMS) are given. Differences between the memory management systems for Versions 1.0 and 2.0 of MIDAS/N are described. Advantages of the improved memory management system are discussed. Chapter 7 describes the input/output system. Chapter 8 contains the physical storage layout of a database. Chapter 9 contains two sample programs. The first sample program is an engineering application program. It shows the efficiency of improved memory management system. The second sample program is an interactive program, which can be used for learning and testing purposes. Finally, discussion on improvement of the database management system is given in Chapter 10. There are appendices to the thesis. They contain the source listing and results of the sample programs.

1.1 Purpose of a Database Management System

Database is a centralized collection of data accessible to several application programs and organized according to a database definition schema (Date, 1975). Database management system (DBMS) is a software handling all access requests and transactions against the database.

As a program increases in size and capabilities, control and information-management activities become relatively more important in relation to productive processing activities. One of the best ways to

attack the problem of interfacing various program components is to communicate between them through a database management system, which accesses databases. Centralized data management has become important in scientific computing during recent years due to increased complexity of problems needing solution.

In using FORTRAN 77 read/write statements, a random access file must have fixed record length. Data must be accessed in terms of records. For accessing first element of each record, one must use "DO LOOP" to read each record to extract first element. Considerable I/O time may be needed if input/output is not properly arranged for efficiency. Usually, if input/output is arranged for efficiency, then the program becomes more complicated and hard to maintain.

When a database management system is used, the database can be designed to contain several data sets. The data sets can be defined with different data models[1] and with different data set dimensions which can be dynamically redefined. Each data model can have several access methods[2]. New access methods can be created to satisfy user's requirements. The efficiency in I/O time is taken care of by the database management system. Therefore, use of a DBMS has following advantages: (i) design of input/output in a program becomes much more flexible, (ii) efficiency in I/O is handled by the DBMS, (iii) the program becomes easy to debug and maintain because more error

[1] The logical organization of the contents of a data set, or schematic representation thereof.

[2] The set of procedures for accessing and transferring data pertaining to a data set.

information is informed by the DBMS, and (iv) large programs can be easily divided into several modules that are connected through a database.

1.2 Overview of MIDAS/N

MIDAS/N is a numerical data oriented database management system. The database of MIDAS/N is stored in a file. It can be direct or sequential access file. The status of a database can be temporary or permanent. Each database contains several data sets. Each data set is defined as a data model which is a one or two dimensional array (vector or matrix). The matrix can be rectangular or triangular. Data type of a data set can be character, short integer, long integer, real, or double precision real. Each data set has several access methods. They are row-wise, column-wise, or submatrix-wise. Any part of a data set can be accessed just by one call statement. The dimensions of a data set can be redefined dynamically.

Usually, in engineering problems, all data cannot fit into main memory. Therefore, they are stored in the secondary memory and transfered into the main memory by parts. In MIDAS/N, the available memory is divided into pages and is called the memory buffer. Data set is also divided into pages of the same size when it is defined. A page of the memory buffer is called a free page if there is no information stored in it. Otherwise, we say it is in use.

MIDAS/N handles all access requests against data sets in terms of pages. Once part of a data set is requested, some page frames must be

assigned to it. When there is no free page, some page must be freed. Page replacement strategy decides the page to be freed. MIDAS/N chooses the least recently used page for replacement. Detail of this strategy are discussed in Section 6.2.

For handling transactions, control information about data sets and page usage must be kept and managed. Control information about data sets is stored in "data set information" table. Control information about page usage is stored in "memory buffer information" table. Both tables are described in Section 6.3.2. New data structures used in MIDAS/N Version 2.0 for keeping and managing these information are introduced in Section 1.4.

1.3 Purpose of the Thesis

For solving large and complicated engineering problems, general purpose programs become large and complex. Program modularization is an important strategy to control its growth. Proper database and a DBMS are needed to communicate data between various modules. Purpose of this thesis is to report design, implementation, and evaluation of a DBMS suitable for engineering applications. Evaluation of a DBMS in engineering applications environment, which is dynamic, is important. It will be discussed in the thesis.

The database management system described in the thesis is MIDAS/N. Version 1.0 of the system was developed by extending the Engineering Data Management System (EDMS; SreekantaMurthy, et al., 1984). The memory management strategies (the storage allocation and page

replacement strategies are explained in Chapter 6), internal data structures and system organization are the same in EDMS and MIDAS/N Version 1.0. However, more capabilities and error trapping was incorporated into MIDAS/N (1.0). It also has more user callable subroutines which gives him more flexibility to manage large amount of data. The main purpose of MIDAS/N (1.0) was to develop a user friendly data management system. The improved capabilities and error trapping are discussed in more detail in Chapter 2.

MIDAS/N (2.0) is an improvement of Version 1.0. The system has been redesigned to enhance its efficiency in terms of memory management as well as computation time. For example in Version 1.0, a free page is found by performing linear search; and the 'least recently used page' is found by using a counter for each page. These linear searches can make the data management system quite inefficient, if the number of pages is large. The data set information record is also duplicated in pages belonging to the same data set. Procedures to overcome these weak points are discussed in Chapter 6. Another improvement of MIDAS/N (2.0) is its organization which is discussed in Section 2.3. The entire system is divided into several segments with clearly defined functions. Each segment can be modified or extended without effecting others. It is believed that this organization will help in maintaining and extending the system.

Some important consideration in redesigning MIDAS/N (1.0) are

1. The system should utilize large as well as small memory efficiently.

2. It should efficiently handle large and small data sets.
3. The system should give flexibility to the user to store the data in one form and retrieve it in another form.
4. Various commonly used data models in scientific applications should be defined by the system.
5. Duplication in overhead information should be avoided.
6. The system should be efficient on personal computer as well as its larger host.

1.4 Some Terminology

1.4.1 Available Stack

Stack is a linear list in which all insertions and deletions are made at one end (refer to Figure 1.1). MIDAS/N (2.0) records free spaces (memory buffer pages or data set information records[3]) using stack. This is called the available stack. Each page or data set information record is called a cell of stack. Each cell has a physical location number called cell number. For the available page stack, the number of cells is equal to the number of memory buffer pages. Similarly, the number of cells for the available data set information record stack is equal to the number of records in "data set information" table.

Stack head pointer records the first available cell number. Each cell records the next available cell number. The last available cell records a particular number, say 0, which means bottom of a stack. If

[3] Records in "data set information" table for storing overhead information (control information) of data sets.

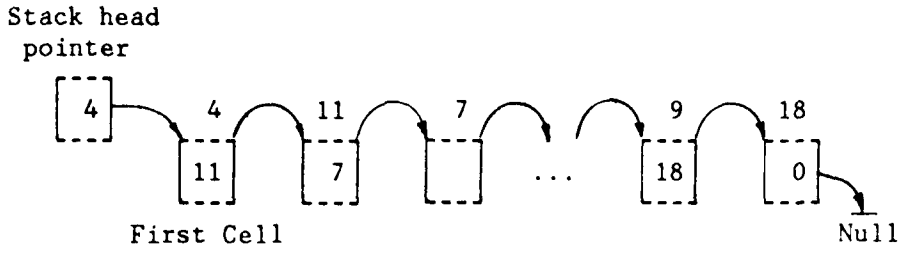


Figure 1.1. Stack Data Structure

stack head pointer contains the particular number (0) then avilable stack is empty.

'Pop out' means that a cell is taken out of the stack. 'Push in' means that a cell is stored into a stack. Using this data structure, finding a free page implies a 'pop out' from available stack. When some page becomes free, it is easy to 'push it in' the available stack.

1.4.2 Least Recently Used (LRU) Double Link

A double link is a linear list for which insertions and deletions can be made at any point of the list. Each cell records its right cell number and the left cell number. A double link is a very useful data structure. With some rules of insertion and deletion, a double link can be used to achieve different functions.

First we use double link to record the least recently used page or the least recently used data set information record. We call this the least recently used double link which has a right pointer and a left pointer (refer to Figure 1.2). A cell still means a page or a data set information record. The rule for LRU double link is that once a cell

is used, it is deleted from current position and inserted to left end. Left pointer records the left end of the link, so it always points to the most recently used cell. Right pointer records the right end of the link which points to the least recently used cell.

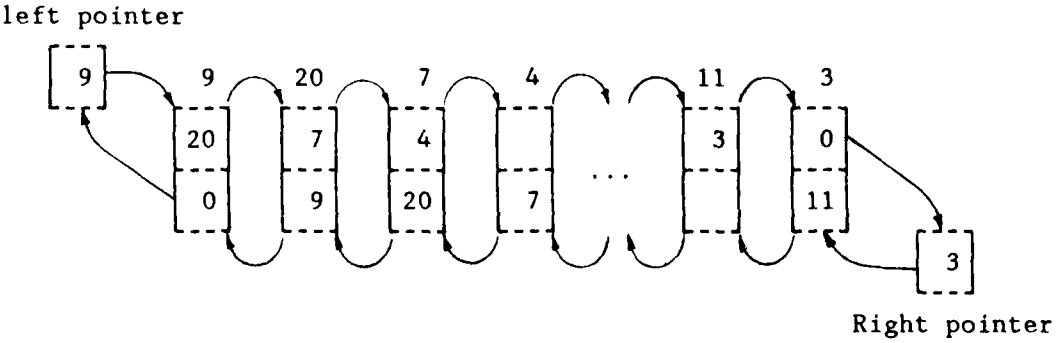


Figure 1.2. Double Link for
Least Recently Used Cell

1.4.3 Connected Page Double Link

The other usage of double link is to record the memory buffer pages connected to a data set; i.e., pages of a data set that are in the main memory. This is called the connected page double link. In this double link, a cell means a page. Connected page double link has only left pointer (refer to Figure 1.3). The rule for connected page double link is that insertion must be made so that logical page number of inserted cell is greater than the left cell and less than the right cell. Therefore, connected page double link is ordered by the logical page number. Left pointer records the smallest logical page number of the double link.

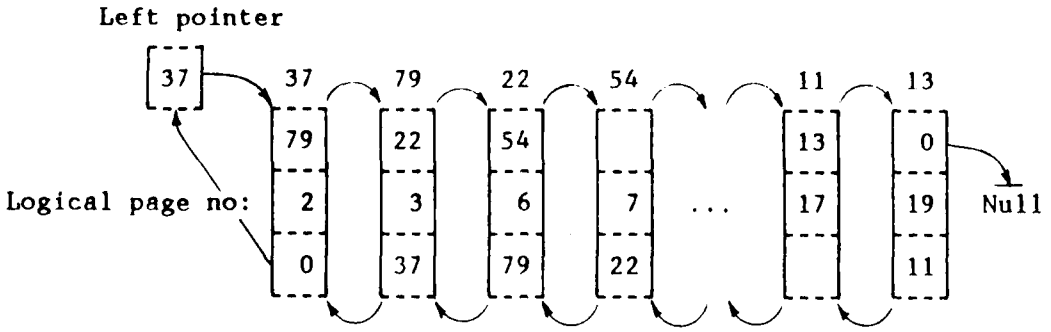


Figure 1.3. Double Link for Pages
Connected to a Data Set

CHAPTER II

IMPORTANT FEATURES OF MIDAS/N

2.1 Important Functions of MIDAS/N

An important consideration in the design of a DBMS is that it must be independent of applications. Therefore, it should be in the form of a library of subroutines that can be called from any application program.

There are two kinds of subroutines in MIDAS/N; database/data set definition and data manipulation. Definition routines are used to create databases and data sets. Data manipulation includes all the other functions, such as open, close, compress, delete, store, retrieve, copy, etc. Function of each subroutine is described in Chapter 3.

Data set is defined as a data model with storage order. The storage order indicates how the data is stored in the secondary memory. It can be row-wise, column-wise, and submatrix-wise. A data set with the row-wise storage order is called a row data set. A row data set can be accessed by any other access method, but it is more efficient to use row-wise access method. That a data set with any storage order can be accessed by all three access methods is one of the improved capabilities of MIDAS/N. This gives more flexibility to the application programmer.

Another improvement in MIDAS/N is the capability to redefine a data set. Dimensions as well as the storage order and the data type of a data set can be dynamically redefined.

2.2 Error Messages

Each subroutine of MIDAS/N checks all the input arguments before performing the requested function. Therefore, the program can give error messages without destroying the existing data. There are two types of error messages: error code number and error information. The error code number from a subroutine is returned to the caller. The application program must check the error code right after the call statement. This makes debugging of an application program easy. Error code remains zero if there is no error in completing the requested function. Explanation of various error codes is given in MIDAS/N user's manual (Shyy, et al., 1984).

The error information is directly displayed on the terminal. It contains three kinds of information: (i) error code number and its meaning, (ii) error related subroutine names, (iii) PRIMOS error information (error information of computer system's utility subroutines), if error is detected by PRIMOS subroutines. Usually, the error information is one of the first two types. The error information from PRIMOS subroutine happens only if there is an error condition which cannot be detected by FORTRAN statement; for example, if the maximum space alloted to the user is exceeded, an error message 'MAXIMUM QUOTA EXCEEDED' is issued by PRIMOS.

2.3 Organization of MIDAS/N

There are four general approaches for organizing a database: hierarchical, network, relational, and numerical. The hierarchical approach provides a nutural way of representing "tree type" relationships. In a hierarchical database, a data object may have many immediate subordinates, but only one immediate superior. On the other hand, network and relational approaches cater to more complex, "many to many" relations between data objects. As such, these models are more appealing for complex databases. Here we will be concerned only with numerical model of data which is in the form of matrices. These matrices can be accessed using various access methods. The organization of database for numerical model is simpler and useful in scientific computing. MIDAS/N (2.0) has improved organization design for dealing with such databases. The system organization is shown in Figure 2.1. Each block is briefly explained in following. Detailed discussions are given in subsequent chapters.

1. Application Program Interface (API):

   This represents the interface between application program and the database management system. This part should be designed so that it does not change in the future, as any change would require changes in application programs. However, we should be able to extend it to add more capabilities.

2. Application Program Interface Processing System (APIPS):

   This is actually the processing part of the application program interface. Its job is to process caller's request for validity

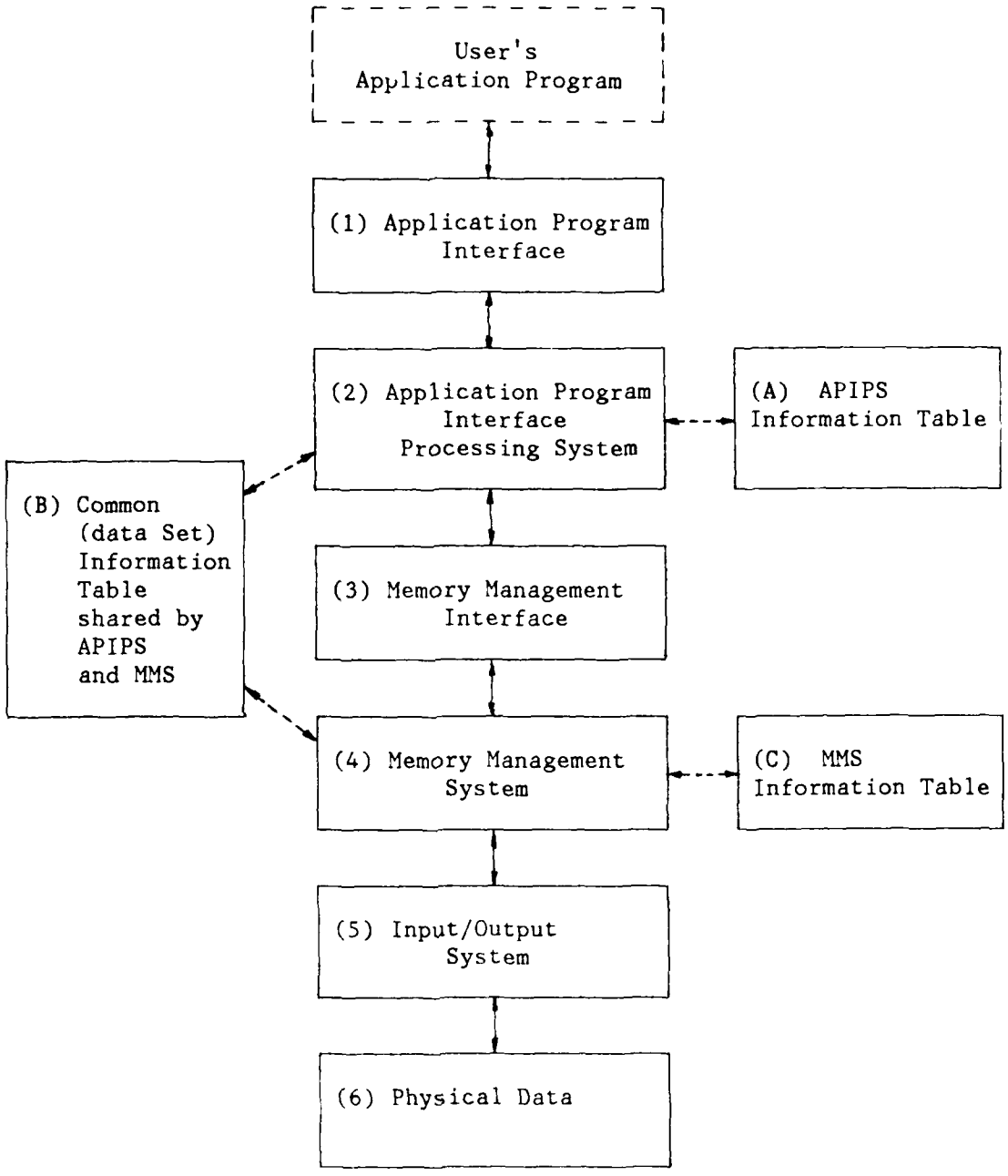


Figure 2.1. System Organization
of MIDSA/N Version 2.0

and correctness. Once this is done, it transfers the request to a sequence of calls to the memory management system.

3. Memory Management Inteface (MMI):

   This is the connection between application program interface processing system and memory management system. Since any modification in this part will cause modification in APIPS and memory management system, it should be designed with care so that it does not need frequent modifications.

4. Memory Management System (MMS):

   Job of the memeoy management system is to transfer data between user buffer and memory buffer, calculate required logical pages, and call I/O system to read-in and update logical pages or information records.

5. Input/Output System (I/O System):

   This part contains a few subroutines to perform disk I/O.

6. Physical Data:

   This is database files which contains control information and actual data on the secondary storage.

Three information tables used by APIPS and MMS are briefly described in following. Detailed definitions are given in Sections 4.3 and 6.3.

A) APIPS Information Table (Database Information Table):

   This table contains information about currently opened databases. It includes database name, status, access type, logical unit number, modification index, and the names and addresses of its data sets. The table is used by APIPS only.

B) Common Information Table (Data Set Information Table):

This table contains data set information of several recently used data sets. It includes data set dimensions, submatrix dimensions, data type, data model, storage order, starting address, total number of words, and left pointer of connected page double link. The table is shared by APIPS and MMS.

C) MMS Information Table (Memory Buffer Information Table):

This table contains information about each page in the memory buffer and its usage status. Page information includes logical page number and modification index. Usage status gives information such as whether it is a free page or a data set connected page. Usage status also indicates whether it is the least recently used page. This information table is used by MMS only.

For studying this organization, we compare it with the organization given by Felippa (1979), described in the following. The database manager is divided into three major parts: (i) data model processor, (ii) resource manager, and (iii) input/output manager. Data model processor processes the user request to define the allocation and access methods which define the mapping of data model onto storage. The resource manager represents the implementation of an allocation method for managing storage resource. The input/output manager handles details of physical access to the stored database.

Organizations of the two systems are similar. Their relationships are as follows:

1. In MIDAS/N (2.0) the application program interface and the memory management interface are isolated into two different segments. The purpose is to emphasize the different roles of the interfaces. Special characters of the two interfaces are given in Chapters 3 and 5.
2. The APIPS of MIDAS/N (2.0) is equivalent to data model processor given by Felippa (1979).
3. The memory management system of MIDAS/N (2.0) is equivalent to resource manager given by Felippa (1979).
4. The input/output system of MIDAS/N (2.0) is equivalent to input/output manager given by Felippa (1979).

Conceptually, MIDAS/N (1.0) contains all the segments of MIDAS/N (2.0), but they are not clearly defined. Finally, we found that a system without proper organization becomes complicated and hard to maintain; especially when it grows.

2.4 How MIDAS/N is Embedded into an Application Program?

MIDAS/N is in the form of a library of subroutines. Application programs can call these subroutines to perform the required functions. When the application program loads it, only the referenced subroutines are loaded. MIDAS/N has a number of user interface routines. More functions will be added and the library will grow in the future. However, design of the system is such that the application program loads only a small part of MIDAS/N. This includes a few routines of APIPS and all the routines of MMS and I/O system. In Chapters 3 to 7

we give subroutines of each segment of MIDAS/N. From there, we can see that subroutines in MMS and I/O system are only a small part of MIDAS/N compared to APIPS routines. It is anticipated that subroutines of MMS and I/O system will not increase due to growth of MIDAS/N. With these features, the application program will not increase too much in size due to use of MIDAS/N. This property becomes extremely important when DBMS is used on personal computers.

In MIDAS/N, dimensions of the memory buffer and over head information records are given by paramaters. We call these system parameters. Changing the system parameters does not influence the organization of a database. This means that a database created by one application program can be used in others without matching their system parameters. With this property, each module of a modularized system can load MIDAS/N with the system parameters which are chosen for its efficiency. For example, the preprocessor of finite element programs can choose smaller page size with more number of pages, since this module has a lot of small data sets. But for the equation solving module, it is better to choose larger page size and less number of pages because it uses a few large data sets. This property is also important for a DBMS to communicate data between a personal computer(or a minicomputer) and its larger host. The reason is the available memory buffer for DBMS in a personal computer is much smaller than that in the host.

CHAPTER III

APPLICATION PROGRAM (USER) INTERFACE

3.1 Objective

Application program interface is a set of subroutines developed to use the database management system in a program. The objective of this segment is to standardize the subroutine calls used by an application programmer. This way when database management system is extended or improved, the existing programs need no modification. This is an important property in a software library. Without this, an existing application program cannot be used with the improved version of the library. In addition, to use an already developed application program, one needs to keep the older version alongwith the newer version. Or, the application program needs to be updated according to the modification in the library. Either way will cause difficulty in developement and maintenance of programs.

The application program interface of MIDAS/N has been carefully designed at the beginning of developement. It does not change in improving from Version 1.0 to Version 2.0, although the memory management system and the internal procedures are completely changed. Due to this reason, the first example program given in Chapter 9 is used with Versions 1.0 and 2.0 without any modification. Therefore, another advantage in standardizing the interface is that one can use

already developed application programs to test the newer version and compare results with older version.

3.2 Application Program Interface Routines

In this section, subroutines used by application programmers are defined to show capabilities of the current version. For more details, MIDAS/N user's manual (Shyy, et al.,1984) should be consulted. All subroutines of application program interface start with the letters 'ND'.

1. NDBCMP : Compresses a database. It collects scattered data sets and stores them together.
2. NDBDEL : Deletes an existing database. Database can be either open or closed.
3. NDBDFN : Creates a new database in user file directory and initializes it.
4. NDBEND : Clears buffer and closes an opened database.
5. NDBINF : Writes all data set names of an indicated database to an output device.
6. NDBOPN : Opens an existing database for read/write operations.
7. NDBRNM : Changes database name; database can be either open or closed.
8. NDGETC : Retrieves some columns of a data set to user's buffer. The data set can have any valid order.
9. NDGETM : Retrieves data set to user's buffer in submatrix order only. The data set must be stored in submatrix order.

10. NDGETR : Retrieves some rows of the data set to user's buffer. The data set can be any valid order.
11. NDINFO : Writes all the database names currently open to an output device.
12. NDJASN : Assigns a job name to data sets. It can be called at any place in the program. All data sets defined after the call will have the new job name concatenated with them.
13. NDPUTC : Stores some columns of data set from user's buffer. The data set can have any valid order.
14. NDPUTM : Stores data set from user's buffer in submatrix order only. The data set must also have submatrix order.
15. NDPUTR : Stores some rows of data set from user's buffer. The data set can have any valid order.
16. NDSARG : Returns data set attributes in arguments.
17. NDSCPY : Copies a data set from the database indicated by NAME1 into the database indicated by NAME2.
18. NDSDEL : Deletes an existing data set.
19. NDSDFN : Defines new data set in an existing database. Data set-job name combination must be unique in a database.
20. NDSGET : Retrieves data set to user's buffer in the specified order. Order of data retrieval can be different from the order of data storage.
21. NDSINF : Writes data set attributes to an output device.
22. NDSPUT : Stores data set from user's buffer in the specified order. The order of putting data into data set can be different from the defined order of the data set.

23. NDSRDF : Redefines a data set without changing the original data. If the original data is not required, NDSDEL and NDSDFN should be used instead of this subroutine. Available options for redefinition are as follows:

    i) Row (column, submatrix) data set can be redefined as any order with the new dimensions.

    ii) Upper triangular data set defined in row order can be redefined as upper triangular row or column order with new dimensions (same applies to lower triangular row and column data sets).

    iii) Upper triangular row (column) data set can be redefined in lower triangular row (column) order with new dimensions. In this case, the upper triangular matrix is transposed to a lower triangular matrix (same applies to lower triangular row and column data sets).

    iv) The data type can be also redefined except for character data type.

24. NDSRNM : Renames an existing data set from DSNAM1 to DSNAM2.

CHAPTER IV

APPLICATION PROGRAM INTERFACE PROCESSING SYSTEM (APIPS)

4.1 Objective

This processing system checks input arguments of the application program interface subroutines and verifies whether or not the function requested is legal. If it is legal, then APIPS proceeds with the requested function by calling its own internal routines and/or by calling the memory management system. Thus this part processes the application program interface and some internal subroutines.

This part uses an APIPS information table (database information table) and refers to a common information table (data set information table) shared by APIPS and MMS. The APIPS information table contains the information about currently opened databases and data sets. This information is used to locate data sets on secondary storage. The common information table contains the attributes of the data sets. This information is used to verify whether or not the function requested is legal. The memory management system (MMS) only contains some basic mapping algorithms to access the stored database (this will be explained in Chapter 6). The APIPS is a preprocessor, translating user's request to some basic methods accepted by the MMS. Therefore, various access methods have been used in the APIPS, and more access methods can be developed, extended, and added.

The internal routines and the APIPS information table are given in following sections. The common information table is given in Section 6.3.2.

4.2 Internal routines of APIPS

This section describes the function of each internal subroutine of APIPS. For more details, MIDAS/N System Reference Manual Version 2.0 (Shyy and Arora, 1984) should be consulted. All internal subroutines of APIPS start with the letters 'NI'.

1. NI1SUB : Intermediate subroutine for NIPUTS and NIGETS; redefines data type of buffer array to deal with character data.
2. NI2SUB : Intermediate subroutine for NIPUTS and NIGETS; redefines data type of buffer array to deal with short integer data.
3. NI4SUB : Intermediate subroutine for NIPUTS and NIGETS; redefines data type of buffer array to deal with real and long integer data.
4. NI8SUB : Intermediate subroutine for NIPUTS and NIGETS; redefines data type of buffer array to deal with double precision real data.
5. NIACOL : Stores and retrieves a data set in column order; data may be stored in any order.
6. NIAROW : Stores and retrieves a data set in row order; data may be stored in any order.

7. NIASUB : Stores and retrieves a data set in submatrix order; for "SUB" data set only.
8. NIBCMP : Internal subroutine collects scattered data sets and stores them together.
9. NICNTN : Concatenates the data set name with the job name.
10. NIDSTD : Deletes the data set information record in the least recently used double link.
11. NIDSTI : Inserts the data set information record in the least recently used double link.
12. NIGETS : Internal subroutine to retrieve data from SUB data set with particular dimensions.
13. NIGLOC : Gets the database location number using database name.
14. NIGLUN : Gets a free logical unit number.
15. NIPUTS : Internal subroutine to store data into SUB data set with particular dimensions.
16. NIRCOL : Redefines row, column or submatrix data set to column data set with new dimensions.
17. NIRDCP : Redefines data set for which data information can be copied directly.
18. NIRDTY : Changes original data type to specified data type for NDSRDF.
19. NIRRDF : Defines a temporary data set with the new dimensions.
20. NIRREP : Deletes the original data set and replaces the temporary data set name by the original data set name.

21. NIRROW : Redefines row, column or submatrix data set to row data set with new dimensions.
22. NIRSUB : Redefines row, column or submatrix data set to submatrix data set with new dimensions.
23. NIRTRI : Redefines the order of triangular date set for NDSRDF.
24. NISACC : Retrieves or stores data set to user's buffer in the specified order.
25. NISCPY : Internal subroutine to copy a data set.
26. NISDEL : Internal subroutine to delete a data set.
27. NISDFN : Internal subroutine to define new data set in an existing database.
28. NISINF : Gets location number of data set information record.
29. NISRCH : Searches for the database and data set location numbers.
30. NISRNM : Internal subroutine to rename a data set.
31. NITRAN : Transfers some data of a temcrary buffer to user's buffer.
32. NIVDTY : Verifies the data type and gets data type code.
33. NIVORD : Verifies the storage order and gets storage order code.
34. NIVRFY : Verifies existence of a data set in the database.

Following subroutines are computer dependent:

35. NIATCH : Attaches to the required subdirectory.
36. NIFDEL : Internal subroutine to delete a closed database file.

37. NIFOPN : Internal subroutine to open a database file.

38. NIFRNM : Internal subroutine to change database file name.

39. NIGPTH : Gets the database name and its pathname from its logical unit number.

4.3 APIPS Information Table

This table is used for storing information about currently opened databases. Therefore, it is also called database information table. The information of each database includes its name, status, access type, logical unit number, modification index, and the names and addresses of its data sets. The names and addresses of data sets are used to locate them in the secondary storage. They are stored in the database when it is closed. The remaining information is used to identify and process databases. Layout of the APIPS information table is given in Figures 4.1 and 4.2. Detailed explanation of various quantities is as follows:

MAXDB : system parameter = 20

: Maximum number of databases that can be opened simultaneously.

MAXDS : system constant = 20

: Maximum number of data sets for each database.

NBBUSY : Integer *4

: Number of databases currently in use.

IBLCUR : Integer *4

: Recently used database location number.

DBNAME : Array [1..MAXDB] of character *20

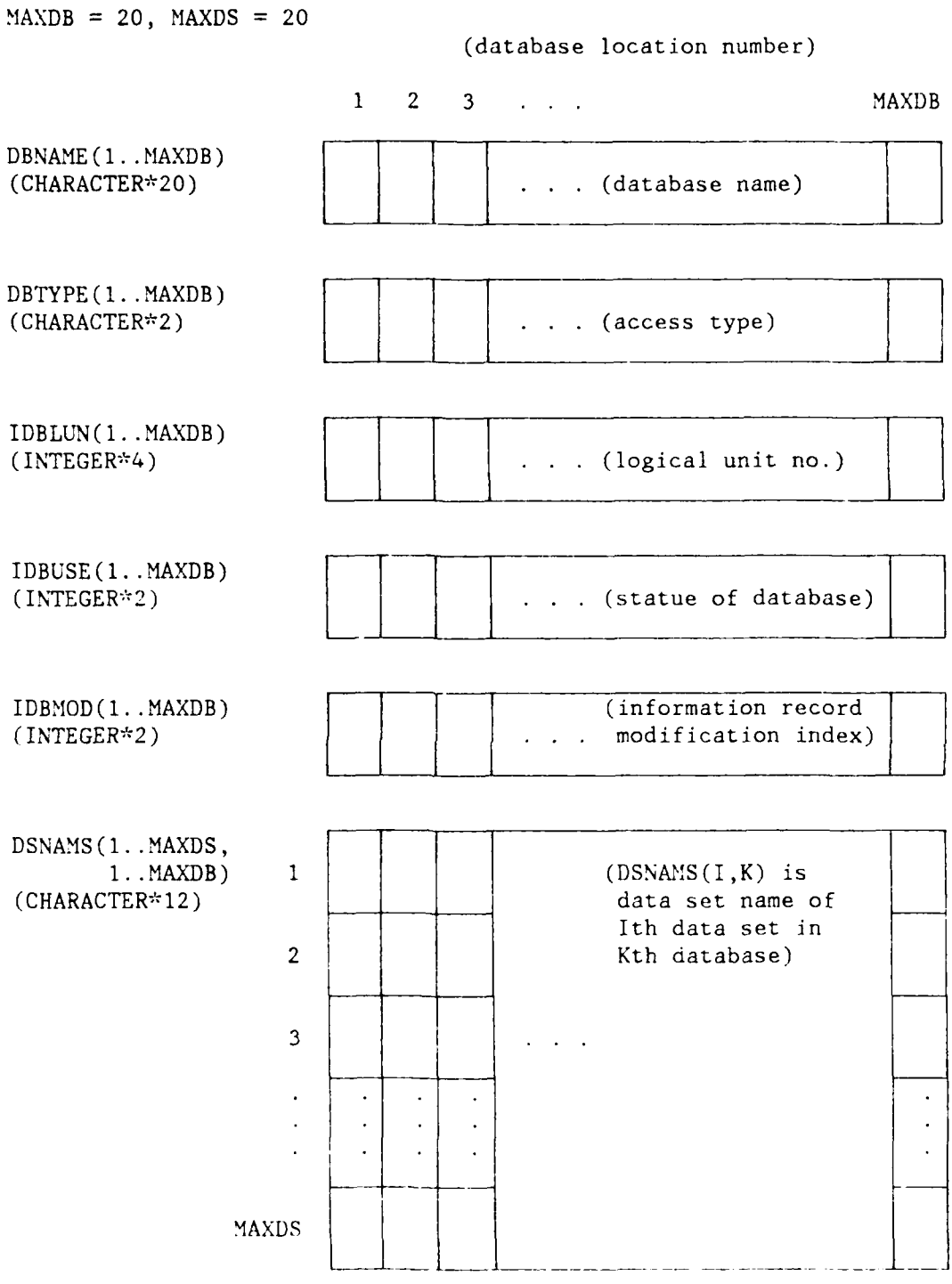


Figure 4.1. Layout of APIPS Information Table (Part 1)

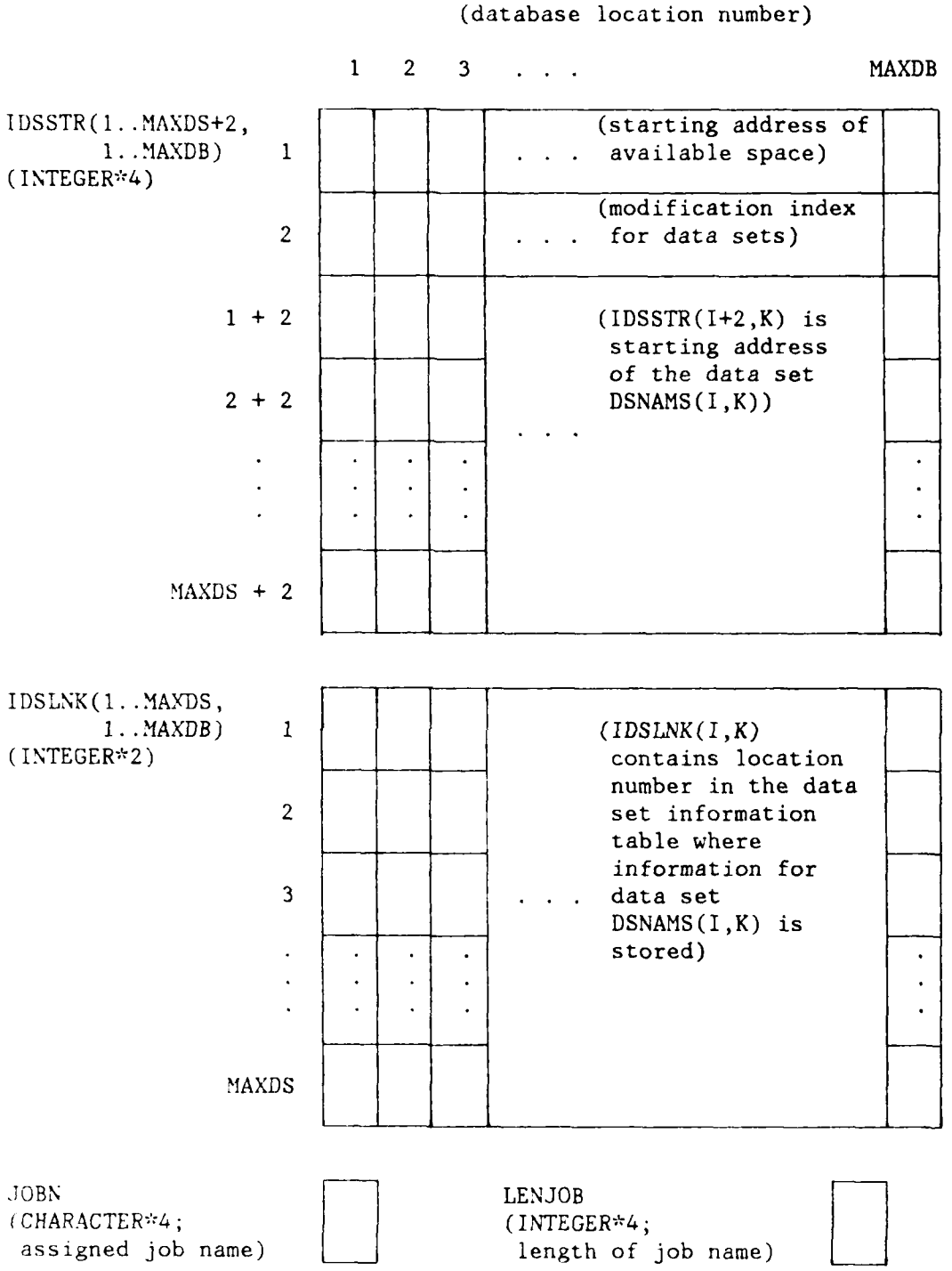


Figure 4.2. Layout of APIPS Information Table (Part 2)

: Database names.

DBTYPE : Array [1..MAXDB] of character *2

: Database access type (Read/Write/Update).

IDBLUN : Array [1..MAXDB] of Integer *4

: Logical unit numbers used by databases.

IDBUSE : Array [1..MAXDB] of Integer *2

: Status of database.

2.. Sequential access permanent database

1.. Direct access permanent database

0.. Not in use

-1.. Direct access temporary database

-2.. Sequential access temporary database

IDBMOD : Array [1..MAXDB] of Integer *2

: Database information modification parameters.

1.. Modified

0.. Not modified

DSNAMS : Array [1..MAXDS, 1..MAXDB] of character *12

: DSNAMS(I,K) is data set name of Ith data set in Kth database.

IDSSTR : Array [1..MAXDS+2, 1..MAXDB] of Integer *4

Row 1 : Starting address of the available space of each database.

Row 2 : Database modification parameter for data sets.

1.. Modified

0.. Not modified

Row 3..MAXDS+2: IDSSTR(I+2,K) is starting address of the data set DSNAMS(I,K).

IDSLNK : Array [1..MAXDS, 1..MAXDB] of Integer *2

: IDSLNK(I,K) contains location number in the data set information table where information for data set DSNAMS(I,K) is stored.

0.. No data set information stored

1..NDSTAB.. Location number of data set information table (for definition of NDSTAB, see Section 6.3.2)

JOBN : Character *4

: Currently assigned job name.

LENJOB : Integer *4

: Length of the job name.

CHAPTER V

MEMORY MANAGEMENT INTERFACE

5.1 Objective

The memory management interface is a set of subroutines used by application program interface processing system (APIPS) to call the memory management system. The memory management interface, like a lower level language used by APIPS, performs the input/output functions. It is designed based on the same concept as the application program interface for user's application program. Therefore, the purpose and importance to standardize the memory management interface is the same as explained in Section 3.1. Each interface routine contains one basic and clearly defined function. The number of routines is kept as small as possible. Keeping one function in each interface routine and using fewer routines can simplify the relationship between APIPS and memory management system. This can help not only in maintenance but also in extending the database management system.

5.2 Memory Management Interface Routines

Subroutines of memory management system interface start with the letters 'NB'. Functions of various subroutines are explained in the following. For more details, MIDAS/N System's Reference Manual Version 2.0 (Shyy and Arora, 1984) should be consulted.

1. NBBINF : Gets database information records from an existing database.
2. NBBUPD : Updates database information records in the database.
3. NBDCRW : Copies data set with different data type.
4. NBDACC : Stores or retrieves data set in the same order in which it is defined.
5. NBDRRW : Directly copies a data set.
6. NBPUPD : Updates data set connected pages into disk.
7. NBSINF : Retrieves data set information records from disk.
8. NBSINT : Initializes a data set to zero.
9. NBSUPD : Updates a data set information record back to disk.

CHAPTER VI

MEMORY MANAGEMENT SYSTEM (MMS)

The self contained memory management system (MMS) is a major part of the database management system (DBMS). It manages the memory buffer and processes input/output requests from application program interface processing system. It is designed to minimize the physical input/output. Strategies used to manage the memory buffer are called memory management strategies. They are storage allocation and page replacement strategies.

6.1 The Storage Allocation Strategy

Since there is limited space for memory buffer, the number of data sets and their sizes are such that they cannot fit into memory buffer. Only parts of some data sets can be stored in memory buffer. The problem then is how to divide the data set into subsets and how to map the subset onto the memory buffer. The method solving this problem is called the storage allocation strategy. There are two popular storage allocation strategies; segmented system and paging system.

Segmented system consists of grouping information into content related subsets and the ability to refer to them. The subset is called a segment. Segments have different lengths depending on the dimension of the data sets. After some I/O between the memory buffer and

secondary storage have been done, the available space in memory buffer becomes discontinuous and each subspace has different size. Now, if some information is to be fetched and several available subspaces can be chosen, then the problem is where to put the information. The method solving this problem is called the placement strategy. The size of the fetched information may not match the size of avilable subspaces. When the information is put into different available subspaces, it produces different smaller subspaces. Sooner or later the available space becomes fragmented into smaller subspaces which are useless. This phenomenon is called storage fragmentation. A segmented system using proper placement strategy can considerably reduce storage fragmentation, but it still exists. Compaction is required which is a method to collect the useless small free spaces into one large contiguous free space. It is not suitable for interactive applications, which is of major interest in engineering. Even with the best compaction routine, the process is extremely inefficient when memory buffer is nearly full. In engineering applications, we like to bring as much information into memory buffer as we can. Therefore, segmented system is not suitable for our purposes.

In the paging system, the memory buffer is divided into equal size blocks called pages. A data set is also divided into same size pages. We will call a page of the memory buffer as a physical page and a page of data set as a logical page. All information transfer between the memory buffer and secondary storage is in terms of pages. A disadvantage of the paging system is that some space may be wasted in

the last page of the data set, which increases when the size of the page increases. This is due to the reason that a data set may not get divided into integral number of pages. Choosing a proper page size will be discussed in Chapter 9. Since data management for engineering applications does not require data to be stored in continuous memory space and paging system requires very small overhead information record, it can be used for large as well as small memory buffers. The paging system is simple and efficient for our application, so is implemented in MIDAS/N.

6.2 The Replacement Strategy

In a paging system, each page has same size. It is unnecessary to consider assignment of free pages (no placement strategy is need). But when there is no free page, some pages must be freed before the required information can be brought into the memory buffer. The question is which page should be replaced first? A method deciding this is called the page replacement strategy. Theoretically, the best choice is to replace a page which is not going to be reused for the longest time. But this is not a practical solution. The best we can do is to make predictions based on the past behavior.

There are several strategies to replace a page, e.g., random, first-in first-out, least frequently used, and least recently used. If we consider the recent behavior to be similar to the near future, then the best strategy is the least recently used one (Lister, 1979). Thus, a very reasonable choice is to select the least recently used page for

replacement. For more details of memory management strategies, one should refer to Fundamentals of Operating System by Lister (1979) or Operating System Concepts by Peterson and Silberschatz (1983).

Both MIDAS/N Version 1.0 and Version 2.0 use the paging system and the least recently used method. The improvement to the memory management system of Version 2.0 is achieved by using a different scheme to record the overhead information. This is given in the next section. Its advantages are discussed in Section 6.4.

6.3 Difference in Two Memory Management Systems

The two memory management systems differ in the organization of the overhead information tables and the way in which information is recorded. In the first two subsections, we describe the overhead information tables and the methods used to record the information for the two memory management systems. Then in the third subsection, we discuss their difference by comparing them.

6.3.1 Overhead Information Table for MMS in MIDAS/N Version 1.0

In MIDAS/N (1.0), there is only one information table called data set information table that is given in MIDAS/N System's Referance Manual Version 1.0 (Shyy, Arora, SreekantaMurthy, and Mukhopadhyay, 1984). Each record in the table corresponds to a physical page in the memory buffer. It records the information of a logical page currently stored in the physical page. It also contains the page usage status and the data set information to which the logical page belongs. The

page usage status includes logical page number in the data set, a counter for keeping relative time of last usage, swap parameter, and starting address in the secondary memory. Data set information recorded includes its name, starting address, ending address, dimensions, storage order, and the data type. Layout of the table is given in Figures 6.1 and 6.2. Explanation of various quantities is as follows:

NPAGES : System parameter = 20

: Total number of physical pages in the memory buffer.

IPSIZ : System parameter = 1024

: Total number of words (2 bytes) in each page.

DSINFO : Array [1..4, 1..NPAGES] of character*12.

IDSINFO : Array [1..12, 1..NPAGES] of integer*4.

ISDSINFO : Array [1..24, 1..NPAGES] of integer*2.

The above three arrays are equivalenced and there explanation follows:

EQUIVALENCE (DSINFO, IDSINFO, ISDSINFO)

Byte

1..12 : Data set name (DSNAME; character*12).

13..16 : Data set starting address (ST; integer*4).

17..20 : Data set ending address+1 (END; integer*4).

21..24 : Empty

25..26 : Number of rows in a submatrix (I1; integer*2).

27..28 : Number of columns in a submatrix (J1; integer*2).

29..30 : Number of rows in the data set (I; integer*2).

31..32 : Number of columns in the data set (J; integer*2).

33..36 : Order of storage (ORDER; character*4).

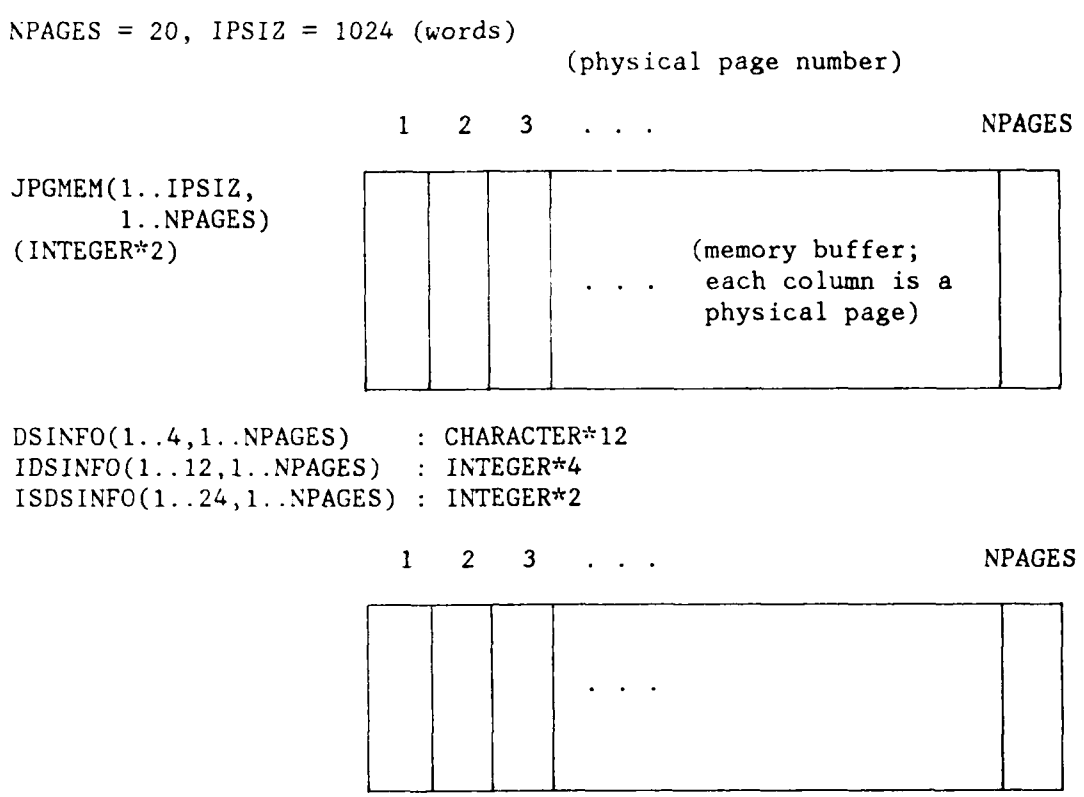
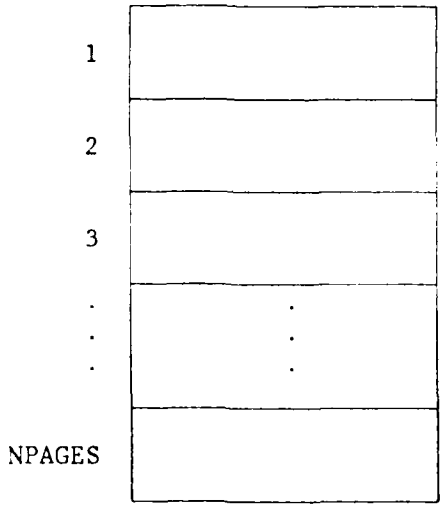


Figure 6.1. Layout of Data Set Information Table for MMS in MIDAS/N Version 1.0 (Part 1)

IPTABLE(1..NPAGES,1..3) : INTEGER*4
ISPTABLE(1..NPAGES,1..6) : INTEGER*2



EQUILALENCE (IPTABLE,ISPTABLE)
each row contains page usage status for a page.

IPTABLE					starting address	
ISPTABLE	logical unit no.	logical page no.	counter	swap param.		

Figure 6.2. Layout of Data Set Information Table for MMS in MIDAS/N Version 1.0 (Part 2)

37..38 : Type of data (DTTYPE; character*4).

41..44 : Starting address of data set elements (ELS; integer*4).

45..48 : Empty.

IPTABLE : Array [1..NPAGES, 1..3] of integer*4.

ISPTABLE : Array [1..NPAGES, 1..6] of integer*2.

EQUIVALENCE (IPTABLE, ISPTABLE)

Byte

1..2 : Logical unit number (integer*2).

3..4 : Logical page number of the data set currently in memory (integer*2).

: Logical page size is same as IPSIZ. Each data set is divided into a number of logical pages.

5..6 : Counter keeping relative time of last use of a page (integer*2).

7..8 : Swap parameter (integer*2).

1..To be swapped back before over writing.

0..Over written without swapping.

9..12 : Starting address of logical page currently in the memory (integer*4).

PGMEM : Array [1..IPSIZ, 1..NPAGES] of integer*2

: Memory buffer (Each column contains a physical page).

NPBUSY : Integer*2.

: Number of pages currently in use.

In MIDAS/N (1.0), the procedure to access some data in a data set can be summarized as follows:

Step 1. By searching through physical pages, record all pages belonging to the data set in a temporary index array. If there is no page belonging to the data set, then get a free page and read in the data set overhead information from secondary memory.

Step 2. Calculate the required logical page numbers by refering to the data set information.

Step 3. Set K as starting logical page number.

Step 4. Find the logical page K by searching through the temporary index array. If the logical page K is not in memory buffer, then get a free page and read in the logical page K from secondary memory.

Step 5. Transfer the data from logical page K to user's buffer.

Step 6. Set page usage status. This requires initialization of the counter of currently used page and addition of one to others.

Step 7. If K equals to the last required logical page number, then stop. Otherwise, add one to K and go to Step 4.

In MIDAS/N (1.0), the procedure to get a free page can be summarized as follows:

Step 1. Find a free page by searching through all physical pages. A free page is recorganized by having a blank data set name. If a free page is found, then stop. Otherwise, creat a free page by using the following steps.

Step 2. Find the least recently used page by comparing counters of all physical pages. The page having the largest counter is the least recently used.

Step 3. Update the information in the least recently used page to make it a free page.

6.3.2 Overhead Information Table for MMS in MIDAS/N Version 2.0

Two information tables, common (data set) and MMS information tables, are used by the MMS in MIDAS/N (2.0).

Common information table, also called the data set information table, contains an available stack, a LRU double link, and a set of data set information records. The number of data set information records is given in system parameter, NDSTAB. The data set information includes data set dimensions, submatrix dimensions, data type, storage order, starting address, total number of words, and a left pointer of connected page double link. The available stack contains the free information records. The LRU double link points out the least recently used data set. Layout of the table is shown in Figures 6.3 and 6.4. Explanation of various quantities is as follows:

NDSTAB : System parameter = 10

: Total columns of data set information table. (Each column is a record, each record contains information of one data set.)

KDSINF : Array [1..8, 1..NDSTAB] of Integer*4.

: Data set information records

: Column access is through data set location number.

Row 1 : Data set starting address (IST).

Row 2 : Total number of words in the data set (NW).

Row 3 : Number of rows in submatrix (ISUB).

Row 4 : Number of columns in submatrix (JSUB).

Row 5 : Number of rows in the data set (NROW).

Row 6 : Number of columns in the data set (NCOL).

Row 7 : Data set storage order code (ICORD).

Row 8 : Data type code for the data set (ICDTY).

KDSAVP : Integer*2

: Available stack head pointer of data set information table.

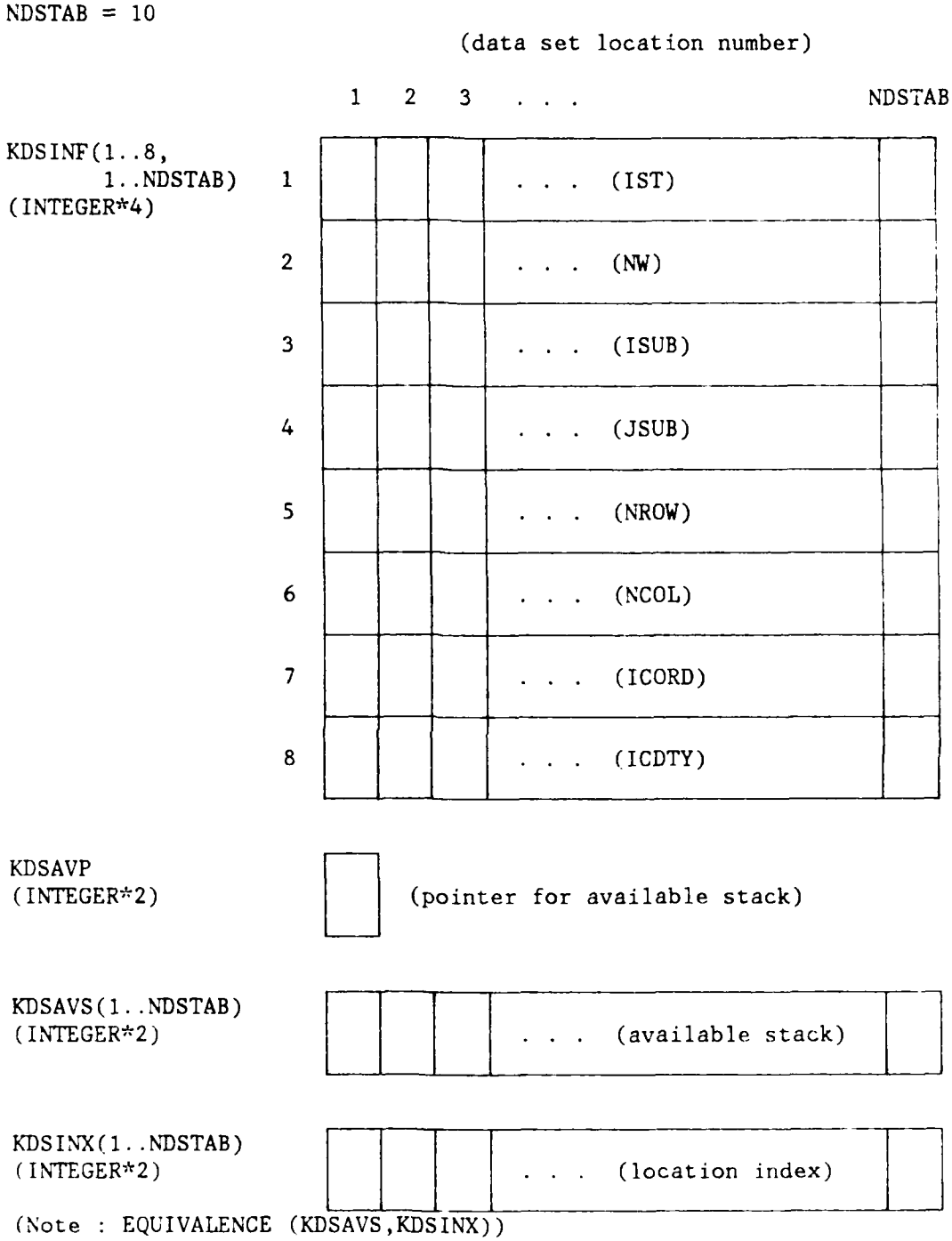


Figure 6.3. Layout of Common Information Table for MMS in MIDAS/N Version 2.0 (Part 1)

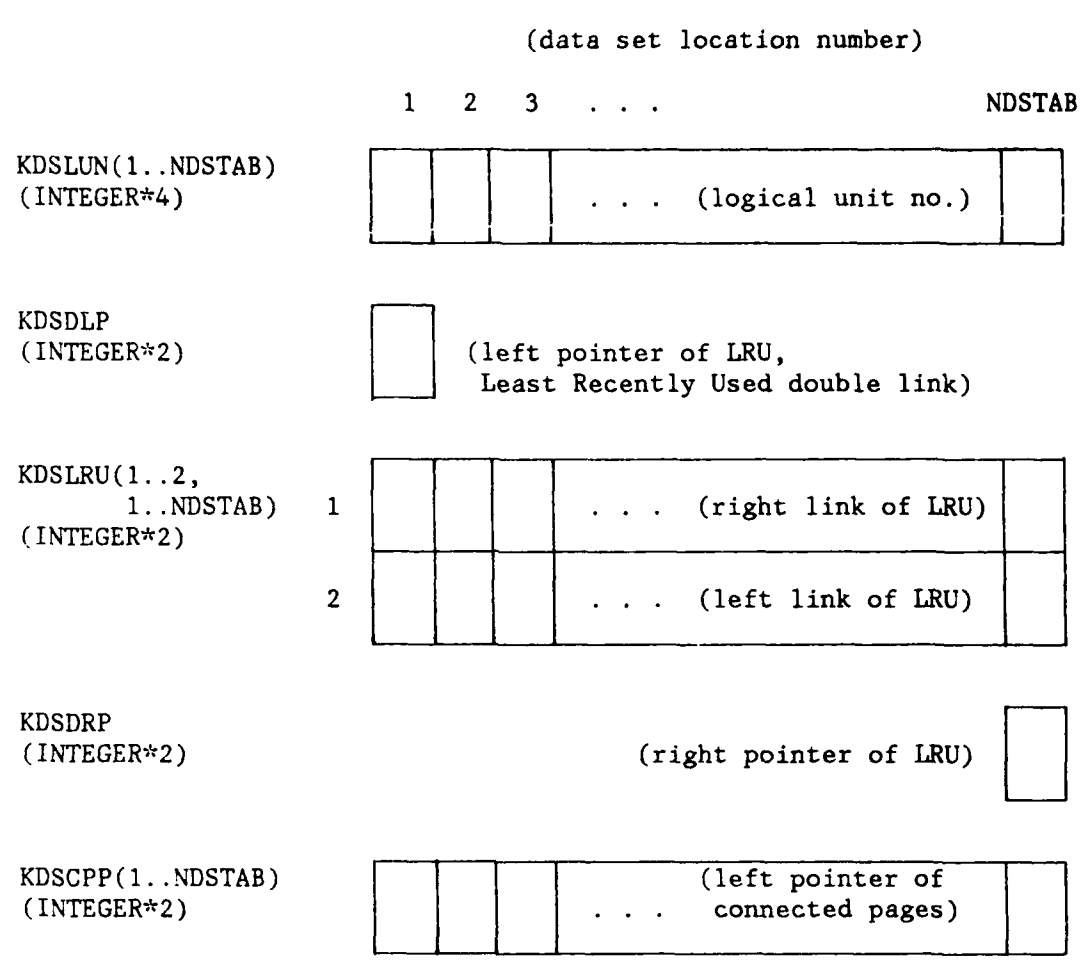


Figure 6.4. Layout of Common Information
Table for MMS in MIDAS/N Version 2.0 (Part 2)

KDSAVS : Array [1..NDSTAB] of Integer*2

: Available stack of data set information table.

KDSINX : Array [1..NDSTAB] of Integer*2

: Index number; records the data set location number in the database information table. (Example: If Kth column of KDSINF stores information for data set DSNAMS(I,J), then KDSINX(K) = (J-1)*MAXDS + I).

KDSLUN : Array [1..NDSTAB] of Integer*4

: Logical unit number of database which contains the data set.

KDSDRP : Integer*2

: Right pointer of LRU double link of data set information table; records least recently used data set.

KDSDLP : Integer*2

: Left pointer of LRU double link of data set information table; records recently used data set.

KDSLRU : Array [1..2, 1..NDSTAB] of Integer*2

: LRU double link of data set information table.

Row 1 : Right link.

Row 2 : Left link.

KDSCPP : Array [1..NDSTAB] of Integer*2

: Left pointer of connected page double link for each data set.

MMS information table contains memory buffer and overhead information. Memory buffer is divided into equal size pages. Overhead information contains information about the data stored in these pages and their usage status. Page information includes logical page number in the data set and modification index. Usage status is indicated by an available stack, a LRU double link, and a set of connected page double links. The available stack records the free pages. The LRU double link records the least recently used as well as most recently used pages. Each data set in the data set information table has a connected page double link. The size of a connected page double link depends on how many pages of the data set are currently stored in the

memory buffer. Each page in memory buffer may belong to one connected page double link only. This table is also called memory buffer information table. Layout of the table is given in Figures 6.5 and 6.6. Explanation of various quantities is as follows:

NPAGES : System parameter = 20

: Number of pages of the memory buffer.

IPSIZE : System parameter = 1024 words (2048 bytes)

: Size of each page (2K bytes)

JPGMEM : Array [1..IPSIZE, 1..NPAGES] of Integer*2

: Memory buffer (Each column contains a physical page).

JPGAVP : Integer*2

: Available stack head pointer of pages of memory buffer.

JPGAVS : Array [1..NPAGES] of Integer*2

: Available stack of pages of memory buffer.

JDSLOC : Array [1..NPAGES] of Integer*2

: Index number; records data set location in data set information table.

JPGDRP : Integer*2

: Right pointer of LRU double link of pages of memory buffer; records least recently used page.

JPGDLP : Integer*2

: Left pointer of LRU double link of pages of memory buffer; records recently used page.

JPGLRU : Array [1..2, 1..NPAGES] of Integer*2

: LRU double link of pages of memory buffer.

Row 1 : Right link.

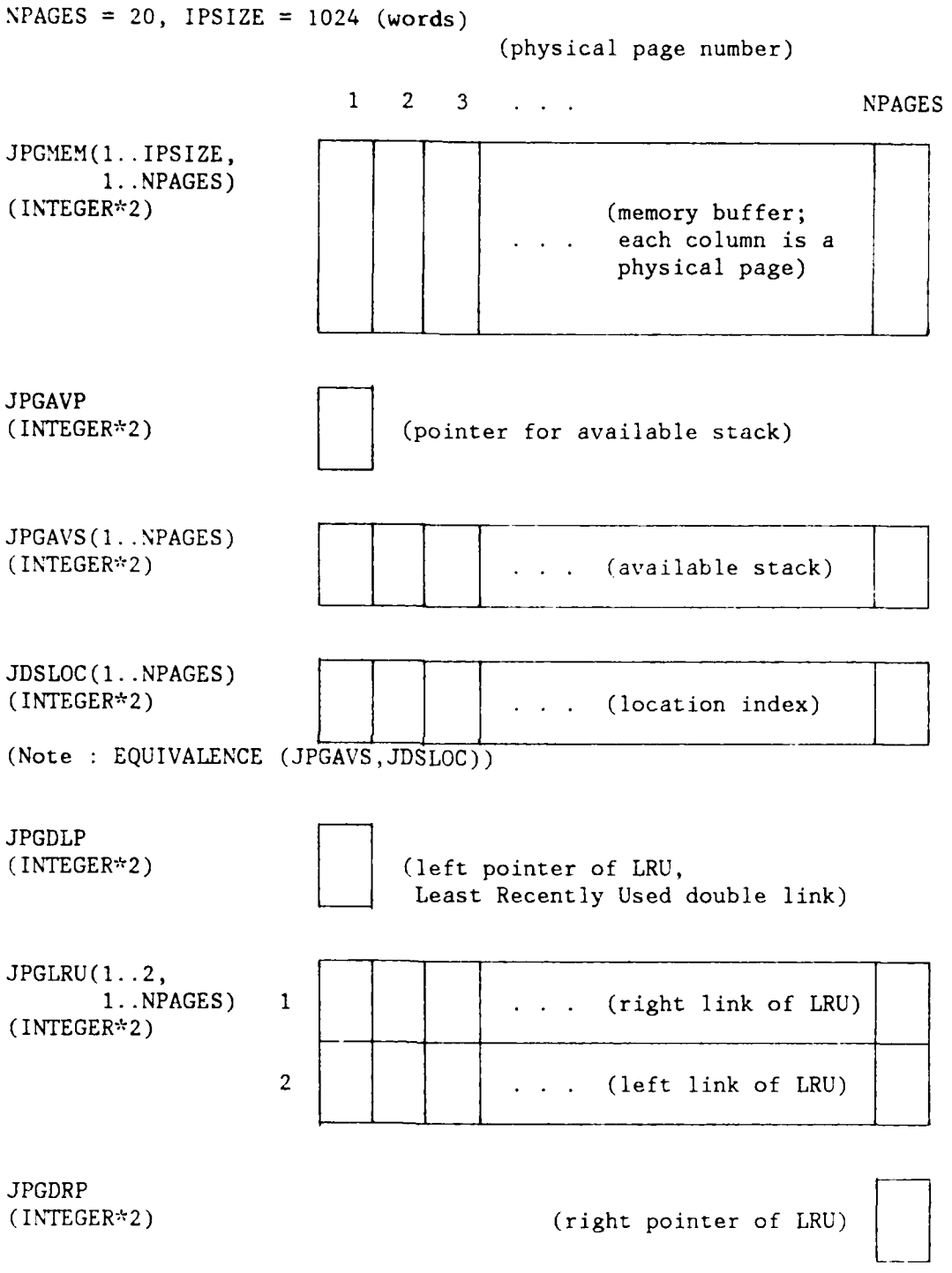


Figure 6.5. Layout of MMS Information Table for MMS in MIDAS/N Version 2.0 (Part 1)

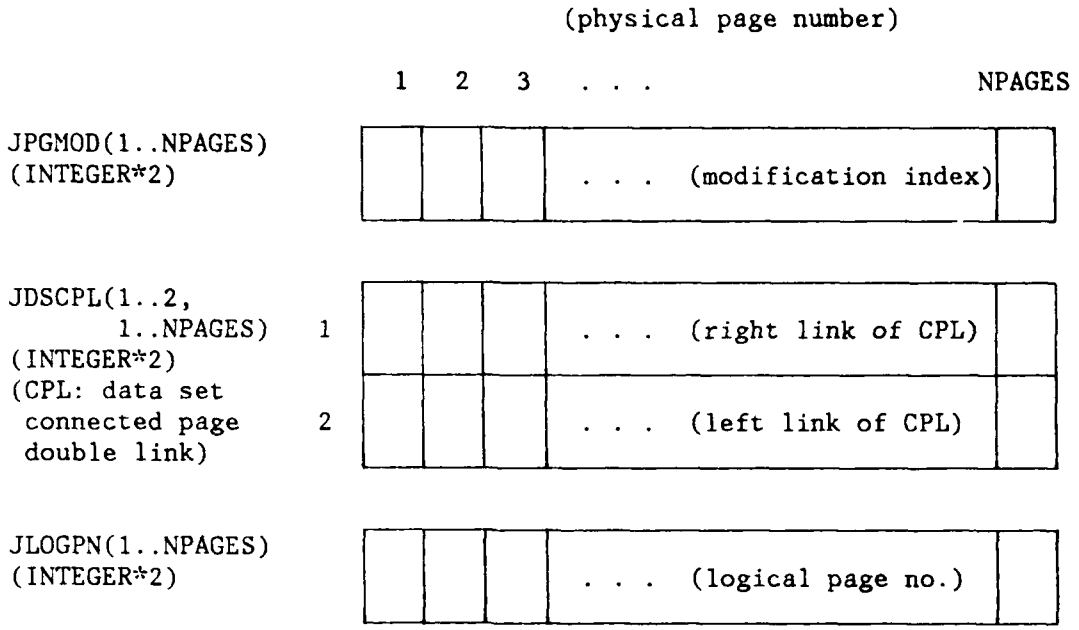


Figure 6.6. Layout of MMS Information Table for MMS in MIDAS/N Version 2.0 (Part 2)

Row 2 : Left link.

JPGMOD : Array [1..NPAGES] of Integer*2

: Page contents modification index.

1.. Modified

0.. Not modified

JDSCPL : Array [1..2, 1..NPAGES] of Integer*2

: Data set connected page double link.

Row 1 : Right link.

Row 2 : Left link.

JLOGPN : Array [1..NPAGES] of Integer*2

: Logical page number of data set connected page double link.

By using these overhead information tables, the procedure to access data from a data set can be summarized as follows:

Step 1. Calculate the required logical page numbers by directly refering to the data set information at the given record number.

Step 2. Set K as starting logical page number.

Step 3. Find the position of the logical page K at the ordered connected page double link by linear search (only once) and store it to pointer I.

Step 4. Check the position at pointer I. If it does not contain the logical page K, then get a free page and read in the logical page K from secondary memory.

Step 5. Transfer the data from logical page K to user's buffer.

Step 6. Set page usage status. This requires to change the position of the currently used page to the left end in the LRU double link.

Step 7. If K equals to the last required page number, then stop. Otherwise, add one to K, change pointer I so that it points to the next position, and go to Step 4.

In MIDAS/N (2.0), the procedure to get a free page can be summarized as follows:

Step 1. Check the available stack. If it is not empty then 'pop out' a free page and stop. Otherwise, creat a free page by using the following steps.

Step 2. Get the least recently used page at the right end of the LRU double link.

Step 3. Update the information in the least recently used page to make it a free page.

6.3.3 Differences Between Two Information Tables

	Version 2.0	Version 1.0
1.	The connection between data sets and pages in memory buffer is formed by connected page double link.	It is formed by recording the data set name and database logical unit number in each page.
2.	All free pages are linked by available stack.	A free page is recognized by having a blank data set name.
3.	The data set information record is stored in data set information table.	It is stored in each page's information record. This is duplicated on pages belonging to the same data set.
4.	The least recently used page is recorded by the right pointer of LRU double link.	It is recorded by a counter in each page. Once a page is used, its counter is initialized to zero and the others' is increased by one. The least recently used page is the one having the largest counter value.

6.4 Efficiency of MMS of MIDAS/N Version 2.0

Advantages of memory management system of MIDAS/N Version 2.0 (called new MMS) in using the data structures defined in Section 1.3 can be summarized as follows:

1. A free page can be directly obtained from the available stack. In Version 1.0, a search is required for the blank data set name in each page's overhead information record.
2. By using least recently used double link it is easy to record least recently used as well as most recently used pages. The new MMS accomplishes this job by transferring six data elements, as explained earlier. This eliminates search. In Version 1.0, use of a page requires updating of all the page counters.
3. By using the least recently used double link, the least recently used page can be directly found from the right pointer of the double link. Version 1.0 requires searching through all pages to find one which has the largest counter.
4. By using the connected page double link, the new MMS always keeps track of the pages currently stored in memory buffer for each data set. Therefore, for an access request, required pages of a data set can be found by searching the first required page in the connected page double link. But, in Version 1.0, the MMS only records data set name on each physical page. Once some data is requested, the MMS searches all the pages belonging to the data set and stores them in a temporary index array. Moreover, the pages stored in index array do not have any order.

To find each requested page requires a linear search in the index array.

All these advantages makes the new system more efficient. In the following subsections, we will look into efficiency from three different aspects.

6.4.1 Handling Large and Small Data Sets in New MMS

Data set size for nodal information may be very small, but for stiffness matrix it may be very large in finite element analysis. Some systems have problems in handling both large and small data sets as it requires large space for overhead information records. Such systems may have an upper bound for data set size, or use search procedures that can be inefficient. A good MMS should be able to handle all data set sizes without above difficulties.

In the new MMS, each data set has a connected page double link to keep track of pages currently stored in the memory buffer. Since size of overhead information record for connected page double link is dependent on the number of pages of the memory buffer and is independent of data set size, there is no wasted information record and there is no limit on data set size. Pages connected in double link are ordered by logical page number. It is easy to delete a page. Also, finding pages does not require a search through all the pages.

A problem in Version 1.0 is that once a large number of pages is used by a large data set then some overhead information about other data sets is lost. The information has to be read in again when those

data sets are used. This increases physical input/output activity. The problem is solved in the new MMS by separating the overhead information for data sets and page buffers.

Due to the above reasons, the new system can efficiently handle large as well as small data sets.

6.4.2 Efficient Use of Small and Large Memory Buffers in New MMS

When a large memory buffer is available, it may be divided into more pages or the page size may be increased. The larger page size increases internal fragmentation which is wastage of memory space due to partially filled pages. It will also increase the total amount of data during input/output if only a small part in a page needs to be accessed.

The increase in number of pages can also result in some inefficiency if proper algorithms are not used. In Version 1.0, the MMS indeed becomes inefficient when the total number of pages increases. This is because it uses linear search to find a required page, and the way it counts and finds a least recently used page. The efficiency of these steps is completely dependent on the total number of pages.

In the new MMS, these problems are solved with the use of proper data structures. First, use of connected page double link to find the first required page is dependent on the length of connected page double link of the data set. This is related to the total number of pages. It takes only one step for eack page to find the remaining pages. This

reduces search drastically. Second, use of least recently used double link to count and find a least recently used page takes only six steps and one step, respectively. Therefore, efficiency of all these steps in the new MMS is almost independent of the number of pages of memory buffer. Under this condition, one can properly choose the page size and the number of pages for the given application and the available memory buffer. An example of how to choose the page size and the number of pages is given in Chapter 9.

Due to above improvements, the new MMS can efficiently use large as well as small memory buffers.

6.4.3 Efficient Use of System Overhead Information in New MMS

Comparing the new MMS with that of Version 1.0, the additional overhead information records used in the former are, available stack, least recently used double link and connected page double link. With this information, the new MMS does not need data set information in each page (as is done in the MMS of Version 1.0). Due to this change, the MMS becomes efficient not only in speed, but also in memory usage due to smaller overhead information. The overhead information for the new MMS is 6+23*NDSTAB+7*NPAGES words and that for Version 1.0 it is 1+30*NPAGES words, where the NDSTAB is the number of data set information records, and NPAGES is the number of pages in memory buffer. Usually, the NPAGES is chosen from 10 to 100 and the NDSTAB is chosen about 5 plus one fourth of NPAGES. As an example, consider NPAGES as 20 and NDSTAB as 10. The overhead information for the new

MMS is 376 words and that for Version 1.0 it is 601 words. If NPAGES is 80 and NDSTAB is 25, the overhead information for the new MMS uses 1141 words and that for Version 1.0 it uses 2401 words.

From the above examples, it seen that the overhead information memory required in the new MMS is reduced to about half of that required in Version 1.0.

6.5 Internal Routines of MMS for MIDAS/N Version 2.0

In this section we will describe the internal routines used in the MMS. They all start with the letter 'NL'. For more details, MIDAS/N System's Reference Manual Version 2.0 (Shyy and Arora, 1984) should be consulted.

1. NL1ASN : Internal subroutine to define data type of buffer array for NLASIN to deal with character data.
2. NL2ASN : Internal subroutine to define data type of buffer array for NLASIN to deal with short integer data.
3. NL4ASN : Internal subroutine to define data type of buffer array for NLASIN to deal with long integer data or real data.
4. NL8ASN : Internal subroutine to define data type of buffer array for NLASIN to deal with double precision real data.
5. NLASIN : Internal subroutine to transfer data between system's buffer and user's buffer.
6. NLCHDT : Changes data type of data from ICDTYO to ICDTYN.
7. NLDINT : Initializes common buffer array according to data type.

8. NLGDAT : Gets one page which contains the specified logical page of data set.

9. NLMVPG : Updates page buffer information back to database.

CHAPTER VII

INPUT/OUTPUT SYSTEM (I/O SYSTEM)

7.1 Objective

This segment contains a few subroutines to perform secondary storage input and output. Usually subroutines in this part call computer system's utility routines for efficiency. Therefore, routines of this segment are usually computer dependent.

In the current I/O System, there are only two subroutines; one for input and the other for output. The data is specified by location address and length, which is the most basic way used in many computers. The purpose of isolating this part is to make the memory management system independent of the computer. This way the database management system can be easily modified for use on other computer systems.

7.2 I/O System Routines

The two subroutines of I/O system start with the letters 'NS'. They both call computer system utilities.

1. NSREAD : Reads data from secondary storage to memory.
2. NSWRIT : Writes data from memory to secondary storage.

CHAPTER VIII

STORAGE LAYOUT

Database files, as defined by the user, are the basic units of storage. Broadly each database contains two types of information:

1. Database control information.
2. Data set information.

Database control information is used for management of the database. It is 164 words long (for MAXDS=20) and consists of the following:

Word

0..119 : Data set names (for maximum 20 data sets).

120..121 : Starting address of next available space.

122..123 : Garbage space index.

0.. no garbage space.

1.. there is garbage space.

124..163 : Starting address of each data set.

Data set information also consists of two parts:

1. Control information (Overhead information).
2. Actual data.

Details of the data set control information is as follows:

Words

1.. 2 : Starting address of actual data of the data set.

3.. 4 : Length of actual data in number of words.

5.. 6 : Number of rows in submatrix.

7.. 8 : Number of columns in submatrix.

9..10 : Number of rows in the data set.

11..12 : Number of columns in the data set.

13..14 : Storage order code.

15..16 : Data type code.

Figure 8.1 shows the database file organization schematically.

DATABASE CONTROL INFORMATION		164 WORDS
DATA SET CONTROL INFORMATION	1st DATA SET	16 WORDS
Actual Data of 1st Data Set		
DATA SET CONTROL INFORMATION	2nd DATA SET	16 WORDS
Actual Data of 2nd Data Set		
. . .		
DATA SET CONTROL INFORMATION	Nth DATA SET	16 WORDS
Actual Data of Nth Data Set		

Figure 8.1.
Database File Structure and
Physical Layout of data

CHAPTER IX

SAMPLE USAGE OF MIDAS/N AND ITS EVALUATION

9.1 Introduction

In this chapter, two sample programs are presented to demonstrate use of MIDAS/N. Each one has a purpose. First, a large linear equation solver shows one use of MIDAS/N in engineering applications. The equation solver program is developed in two ways and is loaded with both versions. Plan of study for this sample program is: (i) describe development of the linear equation solvers, (ii) study the difference between two linear equation solvers which are developed in different ways, (iii) study the effects of changing system parameters, (iv) discribe how to choose system parameters, (v) evaluation of the efficiency of MIDAS/N Versions 1.0 and 2.0, and (vi) discussion of the possible ways to improve MIDAS/N Version 2.0. Each point is discussed in a separate section. The second example is an interactive program. It contains all the functional capabilities of MIDAS/N. It can be used for learning and testing purposes and is given in Section 9.8.

9.2 Development of Linear Equation Solvers

Linear system of equations are encountered in many fields of engineering; especially for finite element techniques. Several methods have been developed for solving linear equations. Most methods have

been developed to efficiently solve the system when the coefficient matrix is stored in the main memory. Once a model (finite element) becomes large and complicated the system of equations becomes larger and it is no longer possible to store the coefficient matrix in main memory. For a general purpose program, equation solver should be able to solve a system of any size which may be stored in the secondary memory. Here we demonstrate the use of MIDAS/N for solving linear equations that are stored on secondary storage. The Choleski decomposition solution technique is used. The coefficient matrix is stored in the skyline form (Bathe, 1982). The equation solver represent applications of utilities of MIDAS/N.

The solution algorithm is developed in two ways. The first way lets the user specify whatever memory space is available to him through subroutine arguments, but the given memory space should be larger than the required minimum working space. The subroutine then utilizes the given space in the best possible manner. This requires complicated programming to efficiently use the available memory. Let this type of programs be designated as Type A. The sample equation solver SKYSOLA is of this type and its source listing is given in Appendix A. The second way is to require the user to provide the minimum working space for the equation solver. The equation solver will use only the minimum working space. This makes the programming task much simpler and efficiency depends on the memory management of the DBMS. Let this type of programs be designated as Type B. The sample equation solver SKYSOLB is of this type and its source listing is given in Appendix B.

Appendix C gives the equation solver SKYSOLC which uses the same method, but the coefficient matrix is stored in the main memory; i.e., there is no secondary I/O. It is designated as Type C. The reason to present two different ways of developing equation solvers with the usage of MIDAS/N is that they are different not only in the programming effort but also in performence and both are important in some sense. This will be discussed later along with their performance data.

Development of equation solver of Type C requires simple coding of the algorithm only. Equation solver of Type B uses the same algorithm, but it calls database routines to read the used columns and write the modified columns in the database. This is done within a DO loop. It uses the minimum working space of fixed size. In equation solver of Type A, data transfer is done outside the solution loop. Each time it reads-in as many columns as possible and processes them. The number of columns which can be read-in is dependent on the user supplied work space. Therefore it requires more sophisticated book keeping and its algorithm is more complicated.

Comparing the equation solvers given in Appendices A, B, and C, the basic differences between the programs with and without the usage of secondary storage can be summarized as follows :

1. A program of Type A or B can solve large problems which cannot be solved by a program of Type C.
2. A program of Type A or B is more complicated compared to Type C. This is due to the reason that the former requires management of the limited work space in addition to coding of the algorithm.

3. Programs of Type A and B are dependent on the database management system. Therefore, they can be considered as its utilities.

9.3 Comparison of Linear Equation solvers of Types A and B

Equation solvers are SKYSOLA for Type A and SKYSOLB for Type B. Both equation solvers are loaded with two versions of MIDAS/N with four different cases[4] of system parameters which are formed by changing the number of pages (NPAGES) and the page size (IPSIZE; Unit: word). The number of calls to MIDAS/N, CPU-time and numbers of reads and writes for solving the system of equations are noted along with dimensions of the system. The number of calls to MIDAS/N is the total number of calls to MIDAS/N routines NDGETR, NDGETC, NDPUTR, and NDPUTC. It is dependent on application program and is invariant of versions and system parameters. The CPU-time is calculated by calling PRIMOS subroutines. Since PRIME operating system is a time-sharing system, the CPU-time is influenced by the number of users. Therefore, all the performance data is collected at the same time[5]. In the following discussion, we assume that the influence of inaccuracy[6] can be neglected. The numbers of reads and writes (Nread and Nwrite) are the

---

[4] Case 1 : NPAGES = 20, IPSIZE = 256;
Case 2 : NPAGES = 80, IPSIZE = 256;
Case 3 : NPAGES = 20, IPSIZE = 1024;
Case 4 : NPAGES = 80, IPSIZE = 1024.

[5] From 2:00 a.m. to 8:00 a.m. at the same day.

[6] In the worst case, the inaccuracy may up to seven percents, but our case is about two percents only.

number of calls to I/O system's routines, NSREAD and NSWRIT, which perform the disk input and output. Dimensions of the system of linear equations are the number of equations (NEQ) and the maximum column height (MAXC) of the coefficient matrix. Complete performance data for equation solvers SKYSOLA and SKYSOLB are given in Appendices D and E respectively. In the following, these performance data will be reorganized or graphed to study the effect of system parameters and to evaluate efficiency of MIDAS/N.

Table 9.1 gives the reorganized data from Appendices D and E for studying the different performance of equation solvers of Types A and B. This table presents data for three different cases. The first two cases use SKYSOLA (Type A) to solve a system of equations with dimensions of 1000(NEQ)x10(MAXC), but Case (a) uses larger work space than Case (b) which uses the minimum work space only. The Case (c) uses SKYSOLB (Type B) to solve the same system of equations and also uses the minimum work space. The performance data of both versions is chosen from Cases 3 of Appendices D and E for system parameters of NPAGES = 20 and IPSIZE = 1024. The difference between the two types can be summarized as follows:

1. A program of Type B is much easier to develop compared to Type A. The program of Type A must contain additional logic and code to full utilize the available work space. Its data transfer depends on the dimension of the work space and its algorithm is somewhat changed from that of Type C. A program of Type B is developed directly from that of Type C without any modification

Table 9.1

Comparison of Equation Solvers of Types A and B

CASE		(a)	(b)	(c)
Data of Application Program	Equation solver	SKYSOLA	SKYSOLA	SKYSOLB
	Program Type	A	A	B
	Minimum Required Work Space (REAL*8)	25	25	25
	Given Work Space (REAL*8)	200	25	25
	Number of Calls to MIDAS/N	843	16270	16262
Performance Data of Version 2.0	Number of Reads	70	70	70
	Number of Writes	30	30	30
	CPU-time (sec)	6.303	33.285	30.690
Performance Data of Version 1.0	Number of Reads	71	71	71
	Number of Writes	30	30	30
	CPU-time (sec)	8.444	75.191	69.011

to the algorithm. This can be seen by comparing subroutines SKYSOLA in Appendix A, SKYSOLB in Appendix B, and SKYSOLC in Appendix C.

2. Equation solver of Type A is considerably more efficient than Type B when the given work space is greater than the minimum work space. With this the number of calls to MIDAS/N routines can be reduced by a factor of more than two when the given work space is doubled. Figure 9.1 shows the number of calls to MIDAS/N vs. work space. In case of using minimum work space, equation solver of Type B is efficient than Type A because more control statements are added in equation solver of Type A and they are useless in this case. This can be seen from Table 9.1.

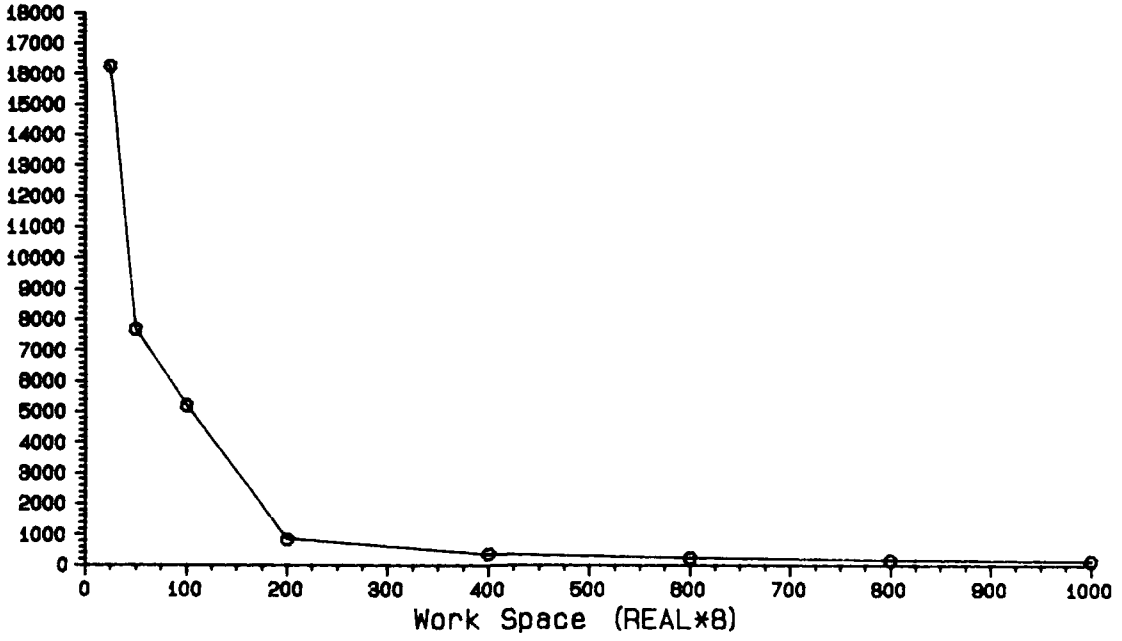


Figure 9.1. Number of Calls to MIDAS/N vs. Work Space

In the above statement, we assume that the inefficiency in CPU-time is due to the increase in number of calls to MIDAS/N routines. This is a reasonable assumption because the Cases (a) and (b) of Table 9.1 are the data of the same equation solver to solve the same system of equations. Both cases have the same numbers of reads and writes. The only difference is the given work spaces which cause the difference in the number of calls to MIDAS/N routines. In each call to MIDAS/N routine, the equation solver needs to reset the pointers and the MIDAS/N does check and locate the data. Both steps take some CPU-time. The calculation of the average CPU-time used in each call to MIDAS/N routine is based on this assumption and will be given in Section 9.6. Although in most cases the equation solver of Type B is inefficient compared to Type A, it does not mean that a program of Type B is useless. Because a simple program, such as Type B, saves time not only in its development but also in its maintenance. These properties are very important for an application program. Once the CPU-time used in each call to MIDAS/N routine is reduced, the program of Type B becomes more important. Therefore, it is needed to introduce both ways of developing a program. The equation solvers given in Appendices A and B show the two typical ways of developing program with the usage of MIDAS/N.

9.4 Effect of The System Parameters

In this section, we will study the effect of the number of pages (NPAGES) and the page size (IPSIZE) by looking at the performance data

of the sample program. Appendix D shows that the performance of SKYSOLA changes alongwith the given work space. To easily compare the difference among the four different cases of system parameters, the performance of SKYSOLA vs. given work space in solving 1000(NEQ) x 10(MAXC) system of equations is drawn in Figures 9.2 to 9.6.

Figures 9.2 to 9.5 show the performance data of Version 2.0. Figure 9.2 shows the CPU-time used. Figure 9.3 shows the number of reads from disk (Nread). Figure 9.4 shows the number of writes into disk (Nwrite). Figure 9.5 shows the total amount of data being transfered from disk which is the total number of reads and writes times the page size. Figure 9.6 shows the CPU-time used by Version 1.0.

The size of memory buffer is the number of pages times page size. The Cases 2 and 3 have the same size of memory buffer. Cases 2 and 3 give information about the effect of changing number of pages with constant memory buffer size. This effect is summarized as follows:

1. Case 3 always uses less CPU-time than Case 2, which shows that for fixed memory buffer larger page size is better in efficiency. This is shown in Figure 9.2.
2. The numbers of reads and writes in Case 3 is about one fourth of that in Case 2, which shows that with constant memory buffer size increasing the page size can reduce the number of I/Os. These are shown in Figure 9.3 and 9.4.

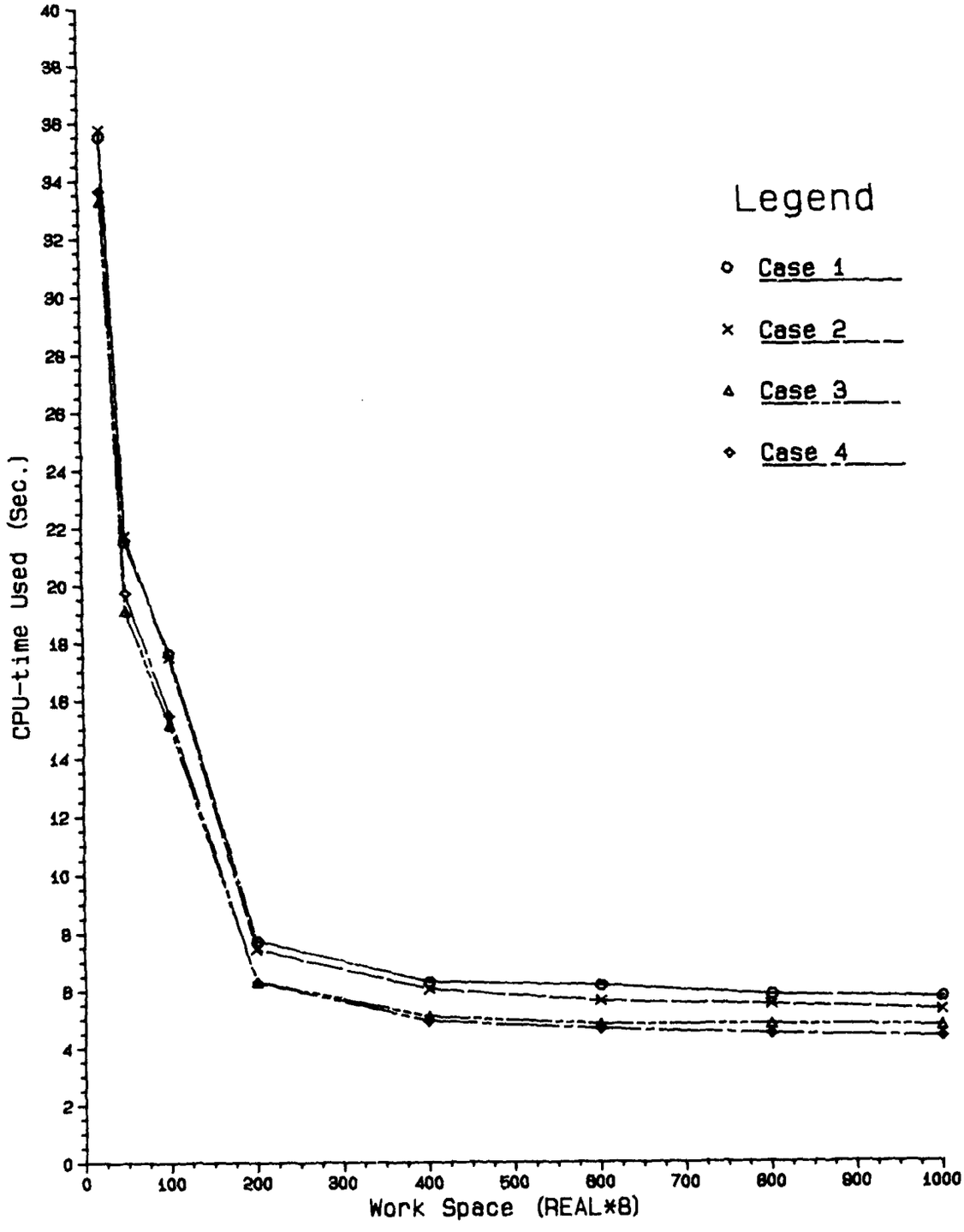


Figure 9.2. CPU-time Used by
Version 2.0 vs. Work Space

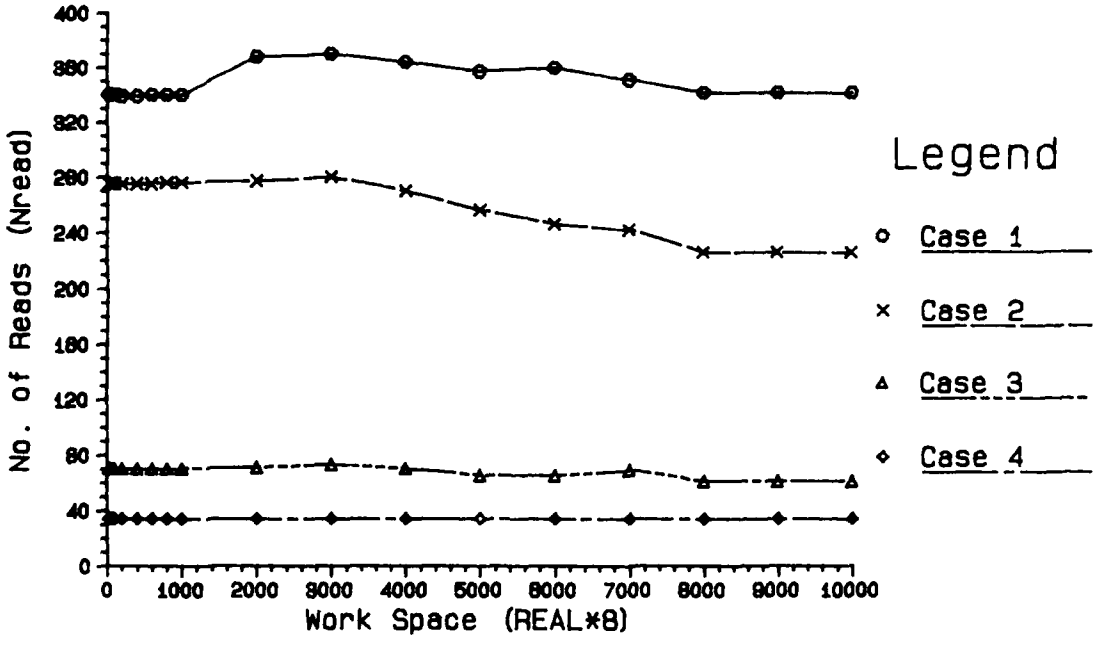


Figure 9.3. Number of Reads (Nread) of Version 2.0 vs. Work Space

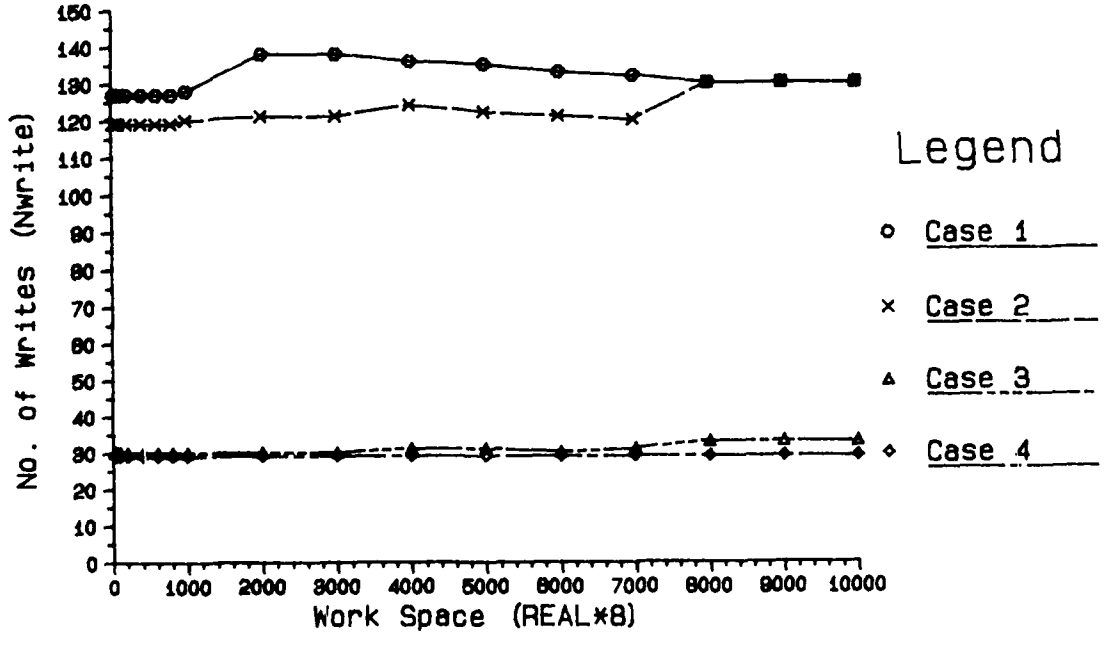


Figure 9.4. Number of Writes (Nwrite) of Version 2.0 vs. Work Space

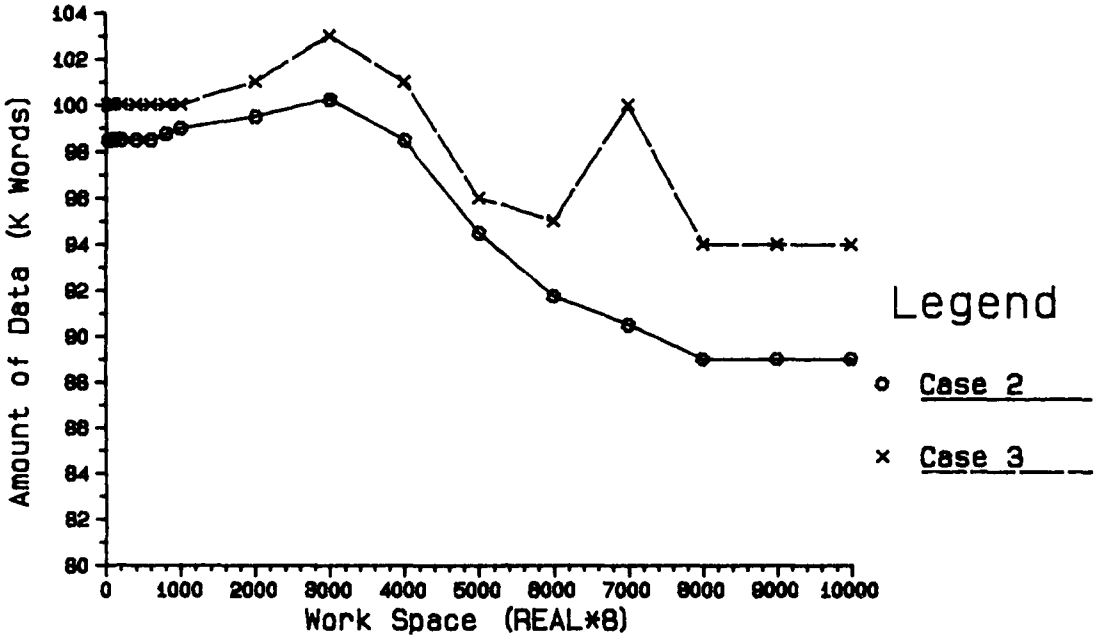


Figure 9.5. Total Amount of Data
Transfered by Version 2.0 vs. Work Space

3. The total amount of data being transfered in Case 3 is always a little more than that in Case 2, as shown in Figure 9.5. This difference is the increase of internal fragmentation due to the increase of page size.

These effect are due to the fact that the equation solver uses only three data sets, one large coefficient matrix and two smaller data sets, and all the data are retrieved in sequence. Therefore, the large page size does not cause much internal fragmentation. In such a case, we can expect the performance with larger page size to be better than that with smaller page size.

The Cases 1 and 2 have the same page size but with the different number of pages. Comparing Cases 1 and 2 gives the information about

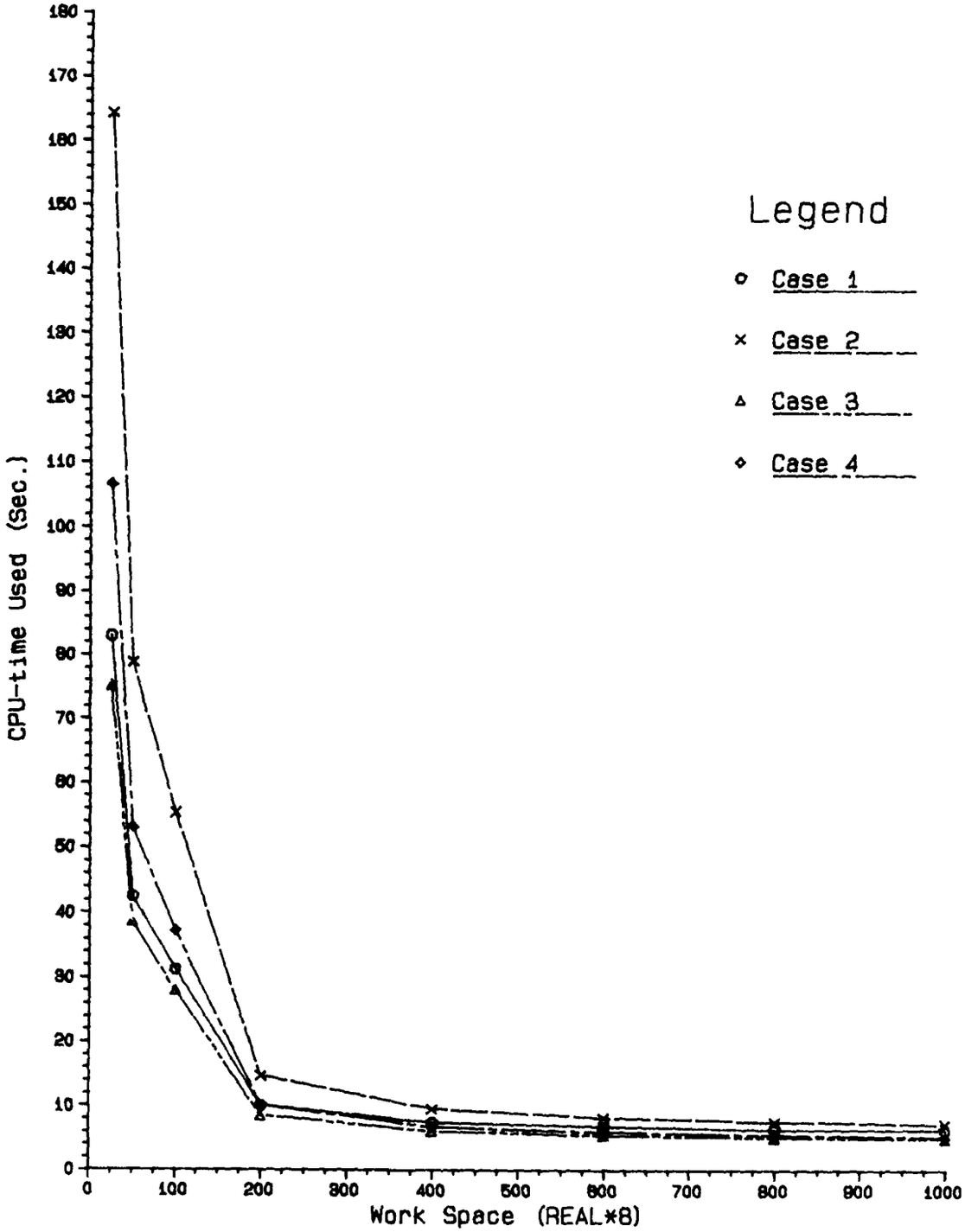


Figure 9.6. CPU-time Used by Version 1.0 vs. Work Space

increasing the memory buffer by increasing the number of pages with constant page size. This is same in Cases 3 and 4. Larger memory buffer can always reduce the disk I/O. Reducing the disk I/O reduces not only the disk I/O time but also some CPU-time. Therefore, using larger memory buffer can save some CPU-time. But, in Version 2.0, it is not true when the given work space is less than 100 (REAL*8), as shown in Figure 9.2. This can be explained as follows. In Version 2.0, efficiency of locating the first retrieved page is dependent on the length of the connected page double link of the data set. In this sample program, the length of connected page double link increases when the number of pages increase. Therefore in Version 2.0 increasing the number of pages causes some inefficiency in locating the data. In case when the given work space is less than 100, the number of calls to MIDAS/N is very large. This drawback in locating the data cancels the advantage gained in reducing the disk I/O.

Since the efficiency of Version 1.0 is completely dependent on the number of pages, as explained in Subsection 6.4.2. We can expect the drawback of increasing the number of pages will becomes severe in Version 1.0. Figure 9.6 indeed shows that in Version 1.0 the efficiency of Case 2 is never better than Case 1. Case 4 will be efficient than Case 3 only when the given work space is larger than 1000.

From previous discussion we know that the sample program benefits from the use of larger page size. Larger memory buffer can reduce the disk I/O. Therefore, in this sample program, it is always efficient to

increase the memory buffer by increasing the page size with constant number of pages. This is shown in Figures 9.2 and 9.6 by comparing the Cases 1 and 3 or 2 and 4.

9.5 How to Choose System Parameters

The system parameters of MIDAS/N Version 2.0 are MAXDB, NDSTAB, NPAGES, ad IPSIZE. MAXDB is the maximum number of databases that can be opened simultaneously. This parameter decides size of the APIPS information table. In MIDAS/N Version 2.0, choosing smaller MAXDB not only saves the overhead information records but also saves the CPU-time to locate a database. Usually in an application program the maximum number of databases which will be opened simultaneously is a fixed number. In such a case, choosing this number to be the MAXDB is the best choice.

NDATAB is the number of the data set information records of the common (data set) information table. It is more complicated to choose NDSTAB than MAXDB. Size of the table depends on NDSTAB. Using smaller NDSTAB saves memory. If NDSTAB is too small, then a data set information record may be removed alongwith some pages of the data set due to another data set information record being read-in, even when there are some free pages in the memory buffer. In such a case, to access the data in these pages requires rereading of both the data set information record and the pages. Therefore, NDSTAB should not be too small. The minimum required NDSTAB is dependent on the number of pages of memory buffer (NPAGES) and the number of data sets being retrieved.

Suggested NDSTAB is greater than 5 plus one fourth of NPAGES and the number of data sets which are accessed in a loop.

NPAGES is the number of pages of the memory buffer. IPSIZE is the size of each page in unit of words (2 bytes). Both belong to MMS information table. The size of memory buffer is NPAGES times IPSIZE. The available memory buffer depends on the application program and the computer. For a fixed memory buffer, how to choose NPAGES and IPSIZE to gain efficiency is of major interest in this section.

From the discussion in the previous section, we see that the choice of the number of pages and page size is dependent on the application program. Here we summarize these effect with some computer dependent effect and give as some principles which can be followed in choosing the number of pages and the page size.

1. Larger page size (implies fewer number of pages) is suitable for a program using a few large data sets and accessing the data in a sequence. The reason that large page size is not suitable for the data sets being retrieved in a random manner is as follows. In such case, once a page is read in for retrieving some data then the remaining data may not be used when it is removed. This phenomenon is called the internal fragmentation.
2. More number of pages is beneficial to a program using more data sets simultaneously or the data in data sets is retrieved randomly. This has the same effect as above.
3. The number of pages should not be too small, even if it satisfies the case of principle 1. For the example of SKYSOLA,

if the total number of pages is three, then the coefficient matrix shares only one page or it will force the other data sets out. This may cause severe internal fragmentation. Therefore, our suggestion is that the number of pages should not be less than 10 or the number of data sets which are retrieved in a loop.

4. For better efficiency, the page size should be same as the size which operating system uses for I/O or a multiple of it (let's call it operating system's I/O size). The reason is that if MIDAS/N page size is smaller than the operating system's I/O size, then each call to read the data may cause the operating system to transfer more data into the memory. In such a case, there may be some inefficiency in the disk I/O time.

According to these priciples, the suggested procedure to choose the number of pages and page size is as follows:

1. Find the operating system's I/O size and decide the memory buffer size. These are dependent on the computer. The memory buffer size should be chosen as a multiple of the operating system's I/O size.
2. Let the page size be the operating system's I/O size to see how many pages the memory buffer should be divided into. Then one of following three steps should be chosen depending on the calculated number of pages.
3. If the number of pages is less than 10, then half the page size until the number of pages is greater than 10. This gives the page size and the number of pages.

4. If the number of pages is greater than 10 and less than 20, then these are the proper number of pages and page size.
5. If the number of pages is greater than 20, then one can choose the page size as any multiple of operating system's I/O size, keeping the number of pages greater than 10.

9.6 Evaluation of the efficiency of MIDAS/N

In this section, we will first calculate the average CPU-time used in each call to MIDAS/N routine, then discuss advantages of the new MMS. Refering Table 9.1, the increased CPU-time from Case (a) to Case(b) in Version 2.0 is 26.982 sec (33.285-6.303). The corresponding increase in the number of calls to MIDAS/N routines is 15427 (16270-843). Following the discussion in Section 9.3, we know that the increased CPU-time is due to the increased number of calls, then each call to MIDAS/N routine takes about 0.00175 sec (26.982/15427). This is in case of MIDAS/N Version 2.0 with NPAGES=20 and IPSIZE=1024. For case of Version 1.0 with same system parameters, the increased CPU-time is 66.747 sec (75.191-8.444; refer to Table 9.1). In this case, the average CPU-time used for each call is 0.00433 sec (66.747/15427[7]). Similar procedure can be used for the other cases of system parameters by using the data from Appendix D. The average CPU-time used for both versions and four cases of system parameters are given in Table 9.2. These average CPU-time also includes the time used by equation solver to set the pointers. It is reasonable to assume that the time used by

---

[7] Note the 15427 is invariant of versions and system parameters.

equation solver is a fixed small amount and will not influence the following discussions because it depends on equation solver only and is invariant of versions and system parameters. Advantages of new memory management system can be seen from this table and are summarized as follows:

Table 9.2

Average CPU-time Used in Calling MIDAS/N Routine

CASE	System Parameters		Average CPU-time Used (sec)	
	NPAGES	IPSIZE (words)	MIDAS/N Version 2.0	MIDAS/N Version 1.0
1	20	256	0.00180	0.00473
2	80	256	0.00184	0.00970
3	20	1024	0.00175	0.00433
4	80	1024	0.00177	0.00625

1. In every case, the average CPU-time used in Version 2.0 is much less than in Version 1.0. This shows efficiency of the improved MMS.
2. In Version 2.0, the average CPU-time used in each call is almost same for all cases, which means that efficiency of Version 2.0 is almost independent of system parameters. With this property,

the new MMS can choose any pair of page size and number of pages in the way explained in Section 9.5 to utilize the available memory buffer for efficiency. Therefore, the new MMS handles large and small data sets efficiently and utilizes small and large memory buffers. In Version 1.0, the average CPU-time used in each call increases tremendously when the number of pages increases. This makes Version 1.0 inefficient in handling small data sets and in using large memory buffer because it is inefficient in using larger number of pages.

3. Refering to Appendices D and E, the number of writes (Nwrite) is the same for Versions 1.0 and 2.0 because both versions use the same memory management strategies. In case of smaller working space, the number of reads (Nread) of Version 1.0 is a little more than that of Version 2.0. This is due to the fact that the data set information record is lost in Version 1.0 when there is no page of that data set in the memory buffer. This record may have to be read-in serveral times. This does not happen in Version 2.0. But, in case of larger working space, the number of reads of Version 1.0 becomes much more than that of Version 2.0. This happens because Version 1.0 reads in the page even when the whole page is going to be modified. It is avoided in Version 2.0. This can be seen clearly in Figure 9.7 which shows the number of reads of both versions in Case 1 of Appendix D.
4. In Version 1.0, the data set dimension is stored in short integer. The maximum possible dimension is 32,767. Therefore,

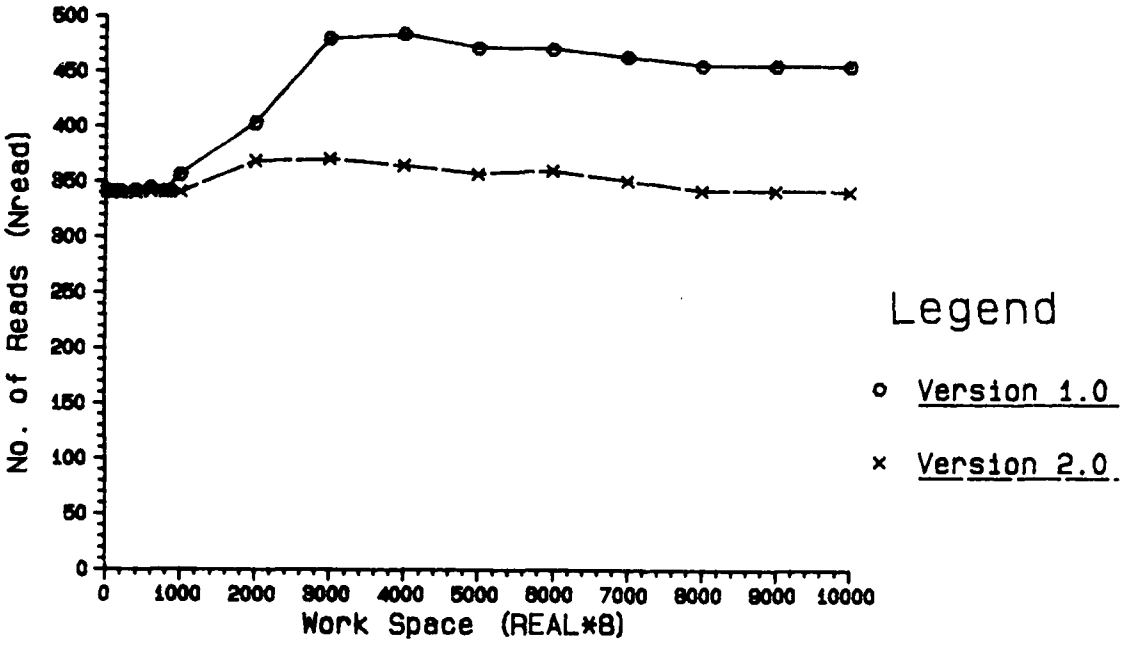


Figure 9.7. Number of Reads (Nread) of Both Versions vs. Work Space

for the present example, program loaded with Version 1.0 cannot solve linear equations of size greater than 1000x50, which are solved by the program of Version 2.0. Because Version 2.0 uses long integer to store the data set dimensions, the maximum possible dimension is 4,294,967,296. Although more words are used to store data set dimensions, the total over head information used in Version 2.0 is less than that in Version 1.0. This is shown in Table 9.3.

The above discussion shows advantages of the new MMS which were stated in Section 6.4.

Table 9.3

The Overhead Information Used in
Memory Management System

CASE	System Parameters		Over Head Information Used in MMS (words)	
	NPAGES	NDSTAB	Version 1.0	Version 2.0
1, 3	20	10	601	376
2, 4	80	20	2401	1026

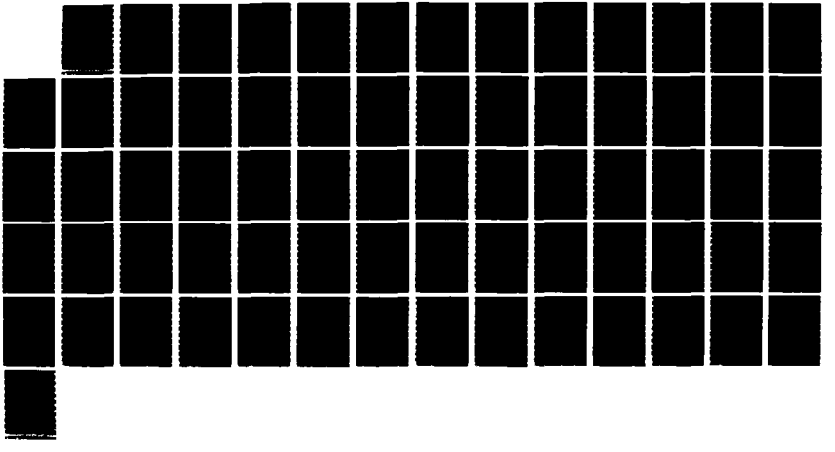
9.7 How to Improve MIDAS/N Version 2.0?

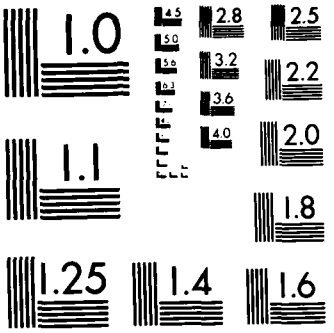
From the previous section we know that the new MMS improves efficiency of MIDAS/N and makes it independent of system parameters. But each call to MIDAS/N routine still takes about 0.0018 second of CPU. Now the question is why does it take so much time and how to improve it?

The time spent in calling MIDAS/N routine can be divided into four parts : (i) to check and locate the requested data set, (ii) to check and locate the accessed data, (iii) to transfer pages of data between memory buffer and secondary memory, and (iv) to transfer the accessed data between memory buffer and user's buffer. From Table 9.1, it is clear that the numbers of reads and writes do not increase due to the increased number of calls to MIDAS/N routines. This means that the physical disk I/O which is part (iii) is already minimized by the MMS.

AD-A174 908 A DATABASE MANAGEMENT SYSTEM FOR ENGINEERING APPLICATIONS(U) IOWA UNIV IOWA CITY OPTIMAL DESIGN LAB Y SHYY ET AL 30 JUN 85 ODL-85 23 AFOSR-TR-86-2205 2/2

UNCLASSIFIED AFOSR-82-0322 F/G 9/2 NL





The time spent in part (iv) depends on the total amount of data to be transfered which cannot be improved by the MMS. To check and locate the accessed data has to be done for accessing the data. Therefore, we cannot save the time spent in part (ii). The only possibility is to save the time spent in part (i). In Version 2.0, the API's routine use two arguments; database and the data set names to identify a given data set. To locate the data set requires search on the database and data set names which is inefficient. For a program as equation solver, there is no need to check and locate the data set each time. Therefore one way to improve the MMS is to avoid the search in locating the data set. This can be done by introducing a data set identification number to identify the data set instead of the original way of using database name and data set name. One interface routine is needed to assign the data set identification number. A new set of accessing routines is required to access the data set by its identification number. Using these interface routines, the time spent in part (i) can be saved.

9.8 Interactive Learning and Testing Program

This is an example interactive part of MIDAS/N Version 2.0. Its main part is written as a subroutine, called NDQURY. By calling this subroutine, the program runs in an interactive mode and gives a list of available commands which is as follows: 1.NDBCMP 2.NDBDEL 3.NDBDFN 4.NDBEND 5.NDBINF 6.NDBOPN 7.NDBRNM 8.NDGETC 9.NDGETM 10.NDGETR 11.NDINFO 12.NDJASN 13.NDPUTC 14.NDPUTM 15.NDPUTR 16.NDSARG 17.NDSCPY 18.NDSDEL 19.NDSDFN 20.NDSGET 21.NDSINF 22.NDSPUT 23.NDSRDF 24.NDSRNM 25.AUTO_M 26.MENU_M 27.QUIT 28.SHOWDS

The first twenty-four commands are exactly the same as the names of subroutines in the API part of MIDAS/N. These commands process the call to the corresponding interface subroutines of MIDAS/N by giving proper arguments. Twenty-fifth command, AUTO_M, sets the program to an AUTO mode. In this mode, the program automatically creats the contents of input array which is used in "PUT" subroutines; NDPUTC, NDPUTM, NDPUTR, NDSPUT. Twenty-sixth command, MANU_M, sets the program to a MANU mode. In this mode, the program asks the data of input array interactively. MENU is the default mode when first time the subroutine NDQURY is entered. Command QUIT is used to exit the interactive mode. The last command, SHOWDS, is used to display the entire data set.

In the following, we discuss some usage of this interactive subroutine. The subroutine NDQURY is compiled and stored in the SMART library, SMART.LIB (Arora and Baenziger, 1984). One can prepare a short program to call the interactive subroutine as shown in Appendix F and loads it with SMART library. Then one can refer to the user's manual of MIDAS/N and interactively run the program to learn each subroutine. This way, a new user can understand each subroutine and its arguments very quickly. Examples of loading the program with SMART library and interactive execution of the program are given in Appendix F.

The interactive program has been used to test MIDAS/N Version 2.0 during its development. Testing is performed by preparing a command input file which contains interactive commands and using this command input file to run the interactive program. The interactive program can

be used to test later versions of MIDAS/N by adding appropriate steps to test the new added functions.

The subroutine NDQURY can be inserted in any part of user's program to check the data stored in data sets at that step. This can be very useful in debugging a program, as it is quite simple to add and delete the call statments without influencing the user's program.

In PRIME computer, another advanced procedure can be used to debug an application program, but this requires the knowledge of using PRIME DEBUG. The procedure is to compile user's program in the DEBUG mode and to load it with subroutine NDQURY which is also compiled with the DEBUG mode. Then by using DBG command to run the program, user can use "CALL" command to invoke subroutine NDQURY at any step of user's program.

The reason to introduce this interactive subroutine is that the data stored in a database is like a black box for the user. Each DBMS has its own database organization which may not be understood by the user and data is stored in a binary form. Without using this interactive subroutine, it is hard to check the data stored in a database; especially data created and used during execution of an application program.

CHAPTER X

DISCUSSION AND CONCLUSION

Purpose of developing MIDAS/N 1.0 and important consideration in redesigning it are discussed. Some useful data structures for improving the MMS are introduced. The important improvement of MIDAS/N 2.0 such as improved flexibility in accessing data, additional capability of dynamically redefining data sets, complete error checking, proper reorganized system organization, and suitability for different size computers are described. Objective of each segment in organization is discussed seperately. The overall idea in designing the new organization is to keep the system easy to extend and improve without influencing the already developed application programs. The related memory management strategies are introduced. Reasons for choosing the paging algorithm and the least recently used method are discussed. Differences between two memory management systems are described. Advantages of the new memory management system are discussed.

The equation solvers in first example give two typical ways to develop application programs with MIDAS/N. The performance of these equation solvers proves the advantages of the new MMS. The most important feature of the new MMS is that its average CPU-time used in calling MIDAS/N routine is independent of the system parameters which

allows the system parameters to be chosen freely to gain efficiency. The possible way to improve the MIDAS/N 2.0 is to develop another set of application interface routines which can locate the data sets easily without performing any search. These can be used when error checking is done in the application programs. The interactive program in the second example shows how to learn and use MIDAS/N. The discussion about the usage of the interactive subroutine NDQURY shows that it is an important tool for debugging application programs in development.

Three important findings of the study are:

1. A good DBMS should avoid any search to locate data set or information records, the least recently used page, and page information or any other records. This is concluded by comparing performance of Versions 1.0 and 2.0 summarized in Table 9.1. Version 1.0 uses search in many steps. It is therefore quite inefficient.
2. The DBMS must be designed so that it can efficiently handle large and small number of pages, memory buffer, and data sets. It can be seen from Table 9.2 that design of Version 1.0 does not satisfy these requirements whereas Version 2.0 meets them.
3. Comparison of CPU-time in Table 9.1 for SKYSOLA, Cases (a) and (b), shows a good strategy that must be followed in developing efficient application programs. Basically the application programs must be developed to efficiently utilize the available memory space specified by the user. The gains in CPU time can be dramatic, 6.303 vs. 33.285 sec for Version 2.0 and 8.444 vs. 75.191 for Version 1.0.

The present study is only a first step in developing a sound database management system for engineering applications. A lot more work needs to be done in this field where database resides in a dynamic and numerically intensive environment.

LIST OF REFERENCES

Arora, J.S. and Baenziger, G., 1984, "SMART: Scientific Database Management and Engineering Analysis Routines: General Description," Optimal Design Laboratory, Department of Civil Engineering, The University of Iowa.

Bathe, K.J., 1982, Finite Element Procedures in Engineering Analysis. Prentice-Hall, Englewood Cliffs, New Jersey.

Date, C.J., 1975, An Introduction to Database Systems. Addison-Wesley, Massachusetts.

Elliott, L., Kunii, H.S., and Browne, J.C., 1978, "A Data Management System For Engineering and Scientific Computing," NASA Conference Publications 2055.

Felippa, C. A., 1979, "Database Management In Scientific Computing-I, General Description," Computers and Structures, Vol. 10, pp. 53-61.

Felippa, C. A., 1980, "Database Management In Scientific Computing-II, Data Structures and Program Architecture," Computers and Structures, Vol. 10, pp. 53-61.

Lister, A.M., 1979, Fundamentals of Operating System. Macmillan, London.

Massena, W. A., 1978, "SDMS - A Scientific Data Management System," NASA Conference Publication 2055.

Peterson, J., and Silberschatz, A., 1983, Operating System Concepts. Addison-Wesley, Massachusetts.

Rajan, S.D. and Bhatti, M.A., 1983, "Data Management in FEM-based Optimization Software," Computers and Structures, Vol. 16, No. 1-4, pp. 317-325.

Shyy, Y.K., and Arora, J.S., 1984, "MIDAS/N System's Reference Manual Version 2.0," Technical Report No. CAD-SS-84.21, Applied-Optimal Design Laboratory, Department of Civil Engineering, The University of Iowa.

Shyy, Y.K., Arora, J.S., Mukhopadhyay, S., and SreekantaMurthy, T., 1984, "MIDAS/N User's Manual," Applied-Optimal Design Laboratory, Department of Civil Engineering, The University of Iowa.

Shyy, Y.K., Arora, J.S., SreekantaMurthy, T., and Mukhopadhyay, S., 1984, "MIDAS/N System's Reference Manual Version 1.0," Technical Report No. CAD-SS-84.13, Applied-Optimal Design Laboratory, Department of Civil Engineering, The University of Iowa.

SreekantaMurthy, T. and Arora, J.S., 1983, "Database Management Concepts In Design Optimization," Technical Report No. CAD-SS-83-13, Design Optimization Laboratory, Division of Materials Engineering, The University of Iowa.

SreekantaMurthy, T., Mukhopadhyay, S., Shyy, Y.K., Arora, J.S., and Reddy, C.P., 1984, "Engineering Database Management System: System's Reference Manual for EDMS," Technical Report No. CAD-SS-84.6, Design Optimization Laboratory, Department of Civil Engineering, The University of Iowa.

APPENDIX A

EQUATION SOLVER OF TYPE A

```
C-----------------------------------------------------------------------
C SMART N-326
C SUBROUTINE SKYSOLA (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
C                     NAMEX,DSNAMX,NE,MAXC,NR,WKSP,NWK,IERR)
C
C Function : Solves the symmetric variable band width system of
C            equations, AX = R.  The input matrix A is stored in
C            one-dimensional form as one-column data set DSNAMA.
C            Input index array I containing indices of diagonal
C            elements locations of matrix A is stored in data set
C            DSNAMI.  Input right hand side column vectors R(NE,NR)
C            are stored in data set DSNAMR.  The input data are
C            undisturbed on return.  The subroutine creates an output
C            data set DSNAMX for storing output column vectors
C            X(NE,NR).  This subroutine utilizes all the woking space
C            in the best possible manner, but the given working space
C            should be greater than minimum requirement.
C
C Arguments: NAMEA  - CHARACTER*(*) - I
C                     name of the database containing the data set
C                     DSNAMA.
C            DSNAMA - CHARACTER*(*) - I
C                     name of the data set containing input symmetric
C                     variable band width matrix A.  The data in the
C                     data set is factorized on return.  The data
C                     set must be a column vector.  But, it can be
C                     row wise or column wise.  Its dimension needs
C                     to be greater than or equal to NT x 1, NT is
C                     total elements number of matrix A.  Data type
C                     can be real or double precision real.
C            NAMEI  - CHARACTER*(*) - I
C                     name of the database to store the data set
C                     DSNAMI.
C            DSNAMI - CHARACTER*(*) - I
C                     name of the data set to store index array
C                     I(NE+1).  The data in the data set is
C                     undisturbed on return.  The data set must be a
C                     column vector with data type of four bytes
C                     integer.  But, it can be row wise or column
C                     wise.  Its dimension needs to be greater than
C                     or equal to NE+1 x 1.
C            NAMER  - CHARACTER*(*) - I
C                     name of the database containing the data set
C                     DSNAMR.
C            DSNAMR - CHARACTER*(*) - I
C                     name of the data set containing right hand side
C                     column vectors R(NE,NR).  The data set should
C                     have same data type as DSNAMU.  Suggested data
C                     set order is row wise.  The data in the data
C                     set is undisturbed on return.
C            NAMEX  - CHARACTER*(*) - I
```

```
C                       name of the database to create the data set
C                       DSNAMX.
C             DSNAMX - CHARACTER*(*) - I
C                       name of the data set to store the solution
C                       column vectors X(NE,NR).  X is solved by using
C                       Choleski solution algorithm.  Data set name is
C                       input.  The data set created is an NE x NR
C                       row data set with data type same as data set
C                       DSNAMR.
C             NE      - INTEGER*4 - I
C                       number of columns of matrix A in full matrix
C                       form.  Default value is that the number of rows
C                       of data set DSNAMI minus one.
C             MAXC    - INTEGER*4 - I
C                       the maximum band width of variable band width
C                       matrix A.  Default value is calculated from
C                       index array I.
C             WKSP    - REAL*4 or REAL*8
C                       working space for the subroutine.  Its data type
C                       needs to be same as data set DSNAMA.
C             NWK     - INTEGER*4 - I
C                       dimension of the working space.  Minimum
C                       requirement is 3MAXC for real data type, 2.5MAXC
C                       for double precision real data type.
C             IERR    - INTEGER*4 - O
C                       error code.  On return,
C                       IERR = 0 means successful solving;
C                       IERR > 0 means singular A matrix; value of IERR
C                       contains the position of singularity;
C                       IERR = -1 means working space is too small;
C                       IERR = -2 means wrong data set dimension or
C                       wrong data type;
C                       IERR = -3 means error message from MIDAS/N.
C
C Note :  i) The solution can be stored under the data set name
C             contained in DSNAMR.  This can be done when NAMEX and
C             DSNAMX are the same as NAMER and DSNAMR.  If this is the
C             case, then DSNAMR must be a row data set.
C-----------------------------------------------------------------------

      SUBROUTINE SKYSOLA (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
     *                    NAMEX,DSNAMX,NE,MAXC,NR,WKSP,NWK,IERR)

      CHARACTER*(*)  NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
     *               NAMEX,DSNAMX
      REAL           WKSP(1)

C-- Check input arguments and data sets

      NER = NE
      MAXCR = MAXC
```

```
      CALL NMVCKI (NAMEI,DSNAMI,NER,MAXCR,NT,WKSP,NWK,IERR)
      IF (IERR.NE.0) GO TO 99

      CALL NMVCKA (NAMEA,DSNAMA,NT,ICDTY,IERR)
      IF (IERR.NE.0) GO TO 99

      IF (ICDTY.EQ.4) THEN
         MINIWK = MAXCR * 3
      ELSE
         MINIWK = MAXCR * 5 / 2
      END IF
      IF (MINIWK.GT.NWK) THEN
         IERR = -1
         GO TO 99
      END IF

      NRR = NR
      CALL NMACRX (NAMEX,DSNAMX,NAMER,DSNAMR,NER,NRR,ICDTY,IERR)
      IF (IERR.NE.0) GO TO 99

C-- Call internal decomposition routine depend on data type

      IF (ICDTY.EQ.4) THEN
         IF (NWK .GT. (NT+NER+1)) THEN
            N2 = NER + 2
            CALL NMVSD2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP(N2),WKSP,NER,IERR)
         ELSE IF (NWK .GT. (MAXCR**2+MAXCR+1)) THEN
            NCOL = (NWK-1) / (MAXCR+1)
            N2 = NCOL*MAXCR + 1
            CALL NMVSD2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP,WKSP(N2),NCOL,IERR)
         ELSE
            NFREE = NWK - MAXCR - (MAXCR+1)
            NCOL = (NFREE + 1) / MAXCR
            N2 = MAXCR + 1
            N3 = N2 + MAXCR*NCOL - 1
            CALL NMVSD4 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP,WKSP(N2),WKSP(N3),NCOL,IERR)
         END IF
         IF (IERR.NE.0) GO TO 99

         IF (NWK .GT. (NT+NER+NER+1)) THEN
            NCOL = NER
            NCOLB = NER
            N2 = NT + 1
            N3 = N2 + NER + 1
         ELSE
            NCOL = (NWK-MAXCR) / (MAXCR+2)
            NCOLB = NCOL
            N2 = NCOL*MAXCR + 1
```

```
            N3 = N2 + NCOL + 1
         END IF
         CALL NMVSF2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,NER,
     *                MAXCR,NRR,WKSP,WKSP(N2),WKSP(N3),NCOL,IERR)
         IF (IERR.NE.0) GO TO 99

         CALL NMVSB2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,NER,
     *                MAXCR,NRR,WKSP,WKSP(N2),WKSP(N3),NCOLB,IERR)

       ELSE
         IF (NWK .GT. (NT+NER/2+1)) THEN
            N2 = (NER/2+1)*2 + 1
            CALL NMVDD2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP(N2),WKSP,NER,IERR)
         ELSE IF (NWK .GT. (MAXCR**2+MAXCR/2+1)) THEN
            NCOL = FLOAT(NWK-1) / (FLOAT(MAXCR)+0.5)
            N2 = NCOL*MAXCR*2 + 1
            CALL NMVDD2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP,WKSP(N2),NCOL,IERR)
         ELSE
            NFREE = NWK - MAXCR - (MAXCR/2+1)
            NCOL = (NFREE+1) / MAXCR
            N2 = MAXCR*2 + 1
            N3 = N2 + (MAXCR*NCOL-1)*2
            CALL NMVDD4 (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                   WKSP,WKSP(N2),WKSP(N3),NCOL,IERR)
         END IF
         IF (IERR.NE.0) GO TO 99

         IF (NWK .GT. (NT+NER+NER/2+1)) THEN
            NCOL = NER
            NCOLB = NER
            N2 = NT*2 + 1
            N3 = N2 + NER*2
         ELSE
            NCOL = FLOAT(NWK-MAXCR) / (FLOAT(MAXCR)+1.5)
            NCOLB = NCOL
            N2 = NCOL*MAXCR*2 + 1
            N3 = N2 + (NCOL+MAXCR-1)*2
         END IF
         CALL NMVDF2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,NER,
     *                MAXCR,NRR,WKSP,WKSP(N3),WKSP(N2),NCOL,IERR)
         IF (IERR.NE.0) GO TO 99

         CALL NMVDB2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,NER,
     *                MAXCR,NRR,WKSP,WKSP(N3),WKSP(N2),NCOLB,IERR)
       END IF

       IF (IERR.EQ.0) RETURN

99     CALL NMIERR (IERR)
```

```
      CALL NMSERR ('SKYSOLA')
      RETURN
      END

**********************************************  SUBROUTINE NMACRB *****

      SUBROUTINE NMACRB (NAME,IERR)

      CHARACTER  NAME*(*)

      LENGTH = LEN(NAME)
      IF (LENGTH.GT.20) LENGTH = 20
      NAME = 'TEMP$DB$OF$MIDASN'

      DO 50 I = 2,LENGTH
         CALL NDBDFN (NAME,' ','RA','TEMP',IERR)
         IF (IERR.EQ.0) THEN
            RETURN
         ELSE IF (IERR.EQ.204) THEN
            NAME(I:I) = 'T'
         ELSE
            GO TO 99
         END IF
50    CONTINUE

99    IERR = -3
      CALL NMSERR ('NMACRB')
      RETURN
      END

**********************************************  SUBROUTINE NMACRX *****

      SUBROUTINE NMACRX (NAMEX,DSNAMX,NAMER,DSNAMR,NE,NR,ICDTY,IERR)

C
C--- Function : Check input data set DSNAMR and creat output data
C---            set DSNAMX with required dimension for forward and
C---            backward substitution of system of equations.
C

      CHARACTER*(*)  NAMEX,DSNAMX,NAMER,DSNAMR,ORDER*4,DTYPE*4

C-- Check data type, storage order,and dimensions of data set DSNAMR

      CALL NDSARG (NAMER,DSNAMR,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IERR)
      IF (IERR.NE.0) GO TO 93

      IF (NE.GT.NROW) THEN
         GO TO 92
```

```
      ELSE IF (ORDER.NE.'ROW' .AND. ORDER.NE.'COL') THEN
         GO TO 92
      ELSE IF (DTYPE.EQ.'REAL') THEN
         IF (ICDTY.NE.4) GO TO 92
      ELSE IF (DTYPE.EQ.'DREA') THEN
         IF (ICDTY.NE.5) GO TO 92
      ELSE
         GO TO 92
      END IF

      IF (NR.EQ.0) THEN
         NR = NCOL
      ELSE IF (NR.GT.NCOL) THEN
         GO TO 92
      END IF

C-- Check whether or not the DSNAMX and DSNAMR are the same data set

      IF (NAMEX.EQ.NAMER .AND. DSNAMX.EQ.DSNAMR) THEN
         RETURN
      END IF

C-- if not, creat output data set DSNAMX

      CALL NDSDUP (NAMER,DSNAMR,NAMEX,DSNAMX,IERR)
      IF (IERR.NE.0) GO TO 93

      CALL NDSRDF (NAMEX,DSNAMX,0,0,'ROW',NE,NR,' ',IERR)
      IF (IERR.EQ.0) RETURN

92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMACRX')
      RETURN
      END


********************************************** SUBROUTINE NMIERR ******

      SUBROUTINE NMIERR (IERR)

C
C--- Function :
C

      IF (IERR.GT.0) THEN
         WRITE (1,2000) IERR
2000     FORMAT (//
     *     ' ERROR - STIFFNESS MATRIX NOT POSITIVE DEFINITE;'/
```

```
     *    ' NONPOSTIVE PIVOT AT EQUATION ',I4/)
      ELSE IF (IERR.EQ.-1) THEN
         WRITE (1,2100)
2100     FORMAT (/
     *    ' ERROR -1 : WORKING SPACE IS TOO SMALL.'/)
      ELSE IF (IERR.EQ.-2) THEN
         WRITE (1,2200)
2200     FORMAT (/
     *    ' ERROR -2 : WRONG DATA SET DIMENSION OR WRONG DATA TYPE.'/)
      ELSE IF (IERR.EQ.-3) THEN
         WRITE (1,2300)
2300     FORMAT (/
     *    ' ERROR -3 : ERROR MESSAGE FROM MIDAS/N.'/)
      ELSE
         WRITE (1,2400) IERR
2400     FORMAT (/
     *    ' NO ERROR MESSAGE AVAILABLE UNDER THIS ERROR CODE :',I5/)
      END IF
      RETURN
      END


************************************************** SUBROUTINE NMSERR *****

      SUBROUTINE NMSERR (SUBNAM)

C
C--- Function : Write error subroutine name to terminal.
C

      CHARACTER  SUBNAM*(*)

      WRITE (1,2000) SUBNAM
2000  FORMAT ('--- the error comes from subroutine "',A,'".'/)
      RETURN
      END


************************************************** SUBROUTINE NMVCKA *****

      SUBROUTINE NMVCKA (NAMEA,DSNAMA,NT,ICDTY,IERR)

C
C--- Function : Check the coeficient matrix data set DSNAMA of the
C---            symmetric variable band width system of equations
C---            and get its data type code.
C

      CHARACTER  NAMEA*(*), DSNAMA*(*), ORDER*4, DTYPE*4

C-- Check data type, storage order, and dimensions of the data set
```

```
      IERR = 0
      CALL NDSARG (NAMEA,DSNAMA,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IERR)
      IF (IERR.NE.0) THEN
         GO TO 93
      ELSE IF (ORDER.NE.'ROW' .AND. ORDER.NE.'COL') THEN
         GO TO 92
      ELSE IF (NT .GT. NROW) THEN
         GO TO 92
      END IF

      IF (DTYPE.EQ.'REAL') THEN
         ICDTY = 4
      ELSE IF (DTYPE.EQ.'DREA') THEN
         ICDTY = 5
      ELSE
         GO TO 92
      END IF
      RETURN

92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMVCKA')
      RETURN
      END


********************************************** SUBROUTINE NMVCKI *****

      SUBROUTINE NMVCKI (NAMEI,DSNAMI,NE,MAXC,NT,IWKSP,NWK,IERR)

C
C--- Function : Check the index array data set DSNAMI and get
C---            total elements number of the coefficient matric.
C

      CHARACTER  NAMEI*(*), DSNAMI*(*), ORDER*4, DTYPE*4
      INTEGER*4  IWKSP(1)

C-- Check data type, storage order, and dimensions of the data set

      IERR = 0
      CALL NDSARG (NAMEI,DSNAMI,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IER2)
      IF (IER2.NE.0) THEN
         GO TO 93
      ELSE IF (DTYPE.NE.'INTL') THEN
         GO TO 92
      ELSE IF (ORDER.NE.'ROW'.AND.ORDER.NE.'COL') THEN
         GO TO 92
```

```
      ELSE IF (NE.EQ.0) THEN
         NE = NROW - 1
      ELSE IF (NE.GE.NROW.OR.NE.LT.0) THEN
         GO TO 92
      END IF

C-- Get the maximum column height and
C-- the total number of elements of the coefficient matrix

      NE1 = NE + 1
      IF (MAXC.EQ.0) THEN
         ILEN = NWK
         IST = 1
5        IEND = IST + ILEN - 1
         IF (IEND.GT.NE1) THEN
            IEND = NE1
            ILEN = IEND - IST + 1
         END IF
         CALL NDGETR (NAMEI,DSNAMI,IST,IEND,1,IWKSP,ILEN,1,IER2)
         IF (IER2.NE.0) GO TO 93
         DO 10 I = ILEN,2,-1
            IDEF = IWKSP(I) - IWKSP(I-1)
            IF (IDEF .GT. MAXC) MAXC = IDEF
10       CONTINUE
         IF (IEND.LT.NE1) THEN
            IST = IEND
            GO TO 5
         END IF
         NT = IWKSP(ILEN) - 1
      ELSE
         CALL NDGETR (NAMEI,DSNAMI,NE1,NE1,1,NT,1,1,IER2)
         IF (IER2.NE.0) GO TO 93
         NT = NT - 1
      END IF

C-- Check the maximum column height and required working space

      IF (MAXC*NE .LT. NT) THEN
         GO TO 92
      ELSE IF (NWK .LT. MAXC*2) THEN
         IERR = -1
         GO TO 99
      END IF
      RETURN

92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMVCKI')
      RETURN
```

```
      END

********************************************** SUBROUTINE NMVCRU ******

      SUBROUTINE NMVCRU (NAMEU,DSNAMU,NAMEA,DSNAMA,NT,IERR)

C
C--- Function : Creat output data set with required dimension
C---            for decomposition of the symmetric variable band
C---            width system equations.
C

      CHARACTER*(*)  NAMEU, DSNAMU, NAMEA, DSNAMA

C-- Check whether or not the DSNAMU and DSNAMA are the same data set

      IF (NAMEU.EQ.NAMEA .AND. DSNAMU.EQ.DSNAMA) RETURN

C-- if not, creat output data set DSNAMU

C     CALL NDSDUP (NAMEA,DSNAMA,NAMEU,DSNAMU,IERR)
      IF (IERR.NE.0) GO TO 93

      CALL NDSRDF (NAMEU,DSNAMU,0,0,'ROW',NT,1,' ',IERR)
      IF (IERR.EQ.0) RETURN

93    IERR = -3
99    CALL NMSERR ('NMVCRU')
      RETURN
      END


********************************************** SUBROUTINE NMVDD2 *****

      SUBROUTINE NMVDD2 (NAMEA,DSNAMA,NAMEI,DSNAMI,NE,MAXC,AA,IA,
     *                   NCOL,IERR)

C
C--- Function : Internal routine controls data transfer and calls
C---            routine "NMVDD3" to decompose a symmetric variable
C---            band width matrix, A = trans(U) * U.  The input
C---            data sets must be checked before calling this
C---            routine.  The decomposed coeficient matrix is stored
C---            in input coefficient matrix on return.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER*(*) NAMEA,DSNAMA,NAMEI,DSNAMI
      DIMENSION     AA(1),IA(1)
```

```
      NCOL1 = NCOL + 1
      IIST = 1
      ISTC = 1

10    IIEND = IIST + NCOL
      IF (IIEND .GT. NE+1) THEN
         IIEND = NE + 1
         NCOL1 = IIEND - IIST + 1
      END IF
      CALL NDGETR (NAMEI,DSNAMI,IIST,IIEND,1,IA,NCOL1,1,IERR)
      IF (IERR.NE.0) GO TO 93

      IASTG = IA(1)
      IASTP = IA(ISTC)
      IAEND   = IA(NCOL1) - 1
      IALEN = IAEND - IASTG + 1
      CALL NDGETR (NAMEA,DSNAMA,IASTG,IAEND,1,AA,IALEN,1,IERR)
      IF (IERR.NE.0) GO TO 93

      NCOL = NCOL1 - 1
      CALL NMVDD3 (AA,IA,ISTC,NCOL,IERR)
      IF (IERR.NE.0) RETURN

      IALEN = IAEND - IASTP + 1
      ISTAA = IASTP - IASTG + 1
      CALL NDPUTR (NAMEA,DSNAMA,IASTP,IAEND,1,AA(ISTAA),IALEN,1,IERR)
      IF (IERR.NE.0) GO TO 93

      IF (IIEND .LT. NE+1) THEN
         IIST = IIST + NCOL - MAXC + 1
         ISTC = MAXC
         GO TO 10
      END IF
      RETURN

93    IERR = -3
      CALL NMSERR ('NMVDD2')
      RETURN
      END


********************************************** SUBROUTINE NMVDD3 *****

      SUBROUTINE NMVDD3 (AA,IA,ISTC,NCOL,IERR)

C
C--- Function : Internal routine to decompose a symmetric variable
C---            band width matrix, A = trans(U) * U.  The input
C---            data sets must be checked before calling this
C---            routine.  The decomposed coeficient matrix is stored
```

```
C---            in input coefficient matrix on return.
C---            This routine is called by "NMVDD2" which does data
C---            transfer.
C---            (for double precision real data type)
C
C--- AA(KL): diagonal element in the Nth column.
C--- AA(KU): highest element in the Nth column.
C--- KH    : height of the Nth column.
C--- KR    : row no. of current element in the Nth column.
C--- KK    : no. of multiplications required in calculating AA(KR,N).
C--- Nth column stands for current column.
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION     AA(1),IA(1)
      DATA          ZERO/0.D0/,AMIN/1.D-30/

C-- reset the index array IA.

      IDELTA = IA(1) - 1
      DO 10 I = 1,NCOL+1
         IA(I) = IA(I) - IDELTA
10    CONTINUE

C-- decomposition

      IERR = 0
      DO 50 N = ISTC,NCOL
         KL = IA(N)
         KU = IA(N+1) - 1
         KH = KU - KL
         KR = N - KH
         KLT = KU
         IC = 0

C-- loop 30 generates off-diagonal elements in the Nth column.

         DO 30,J = 1,KH
            KI = IA(KR)
            ND = IA(KR+1) - KI - 1
            KK = MIN (IC,ND)
            C = ZERO
            DO 20,L = 1,KK
               C = C + AA(KI+L)*AA(KLT+L)
20          CONTINUE
            AA(KLT) = (AA(KLT) - C) / AA(KI)
            IC = IC + 1
            KR = KR + 1
            KLT = KLT - 1
30       CONTINUE
```

```
C-- generates diagonal element in Nth column.

         C = ZERO
         DO 40 JJ = 1,KH
            C = C + AA(KL+JJ)**2
40       CONTINUE
         AA(KL) = AA(KL) - C
         IF (AA(KL).LT.AMIN) THEN
            IERR = N
            GO TO 99
         END IF
         AA(KL) = SQRT(AA(KL))
50     CONTINUE
       RETURN

99     CALL NMSERR ('NMVDD3')
       RETURN
       END


********************************************** SUBROUTINE NMVDD4 *****

       SUBROUTINE NMVDD4 (NAMEA,DSNAMA,NAMEI,DSNAMI,NE,MAXC,
      *                   AA,BB,IA,NCOL,IERR)

C
C--- Function : Internal routine to decompose a symmetric variable
C---            band width matrix, A = trans(U) * U.  The input
C---            data sets must be checked before calling this
C---            routine.  The decomposed coeficient matrix is stored
C---            in input coefficient matrix on return.
C---            (for double precision real data type)
C

       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
       CHARACTER*(*) NAMEA,DSNAMA,NAMEI,DSNAMI
       DIMENSION     AA(1),BB(1),IA(1)
       DATA          ZERO/0.D0/,AMIN/1.D-30/

       MAXB = NCOL*MAXC - 1
       MAXI  = MAXC - 1
       MAXC1 = MAXC + 1
       CALL NDGETR (NAMEI,DSNAMI,1,1,1,IA(MAXC1),1,1,IERR)
       IF (IERR.NE.0) GO TO 93

       DO 80 N = 1,NE
          N1 = N + 1
          DO 10 I = 1,MAXC
             IA(I) = IA(I+1)
10        CONTINUE
```

```
      CALL NDGETR (NAMEI,DSNAMI,N1,N1,1,IA(MAXC1),1,1,IERR)
      IF (IERR.NE.0) GO TO 93

      KNL = IA(MAXC)
      KNU = IA(MAXC1) - 1
      KNH = KNU - KNL
      CALL NDGETR (NAMEA,DSNAMA,KNL,KNU,1,AA,MAXC,1,IERR)
      IF (IERR.NE.0) GO TO 93

      KNP = KNH + 1
      KIAC = MAXC - KNH
      IC = 0
      DO 50 I = 1,KNH,NCOL
         IBNCOL = MIN (NCOL, KNH-I+1)
         IBS = IA(KIAC)
         IBE = IA(KIAC+IBNCOL) - 1
         IBL = IBE - IBS + 1
         IF (IBL.GT.MAXB) THEN
            IBL = MAXB
            IBE = IBE - 1
         END IF

         CALL NDGETR (NAMEA,DSNAMA,IBS,IBE,1,BB,IBL,1,IERR)
         IF (IERR.NE.0) GO TO 93

         IBIDX = 1
         DO 40 II = 1,IBNCOL
            KIL = IA(KIAC)
            KIH = IA(KIAC+1) - KIL - 1
            KK  = MIN (IC,KIH)

            C = ZERO
            DO 20 J = 1,KK
               C = C + AA(KNP+J)*BB(IBIDX+J)
20          CONTINUE

            AA(KNP) = (AA(KNP)-C) / BB(IBIDX)
            IC = IC + 1
            KNP = KNP - 1
            KIAC = KIAC + 1
            IBIDX = IBIDX + KIH + 1
40       CONTINUE
50    CONTINUE

      C = ZERO
      DO 60 J = 1,KNH
         C = C + AA(1+J)**2
60    CONTINUE

      AA(1) = AA(1) - C
      IF (AA(1).LT.AMIN) THEN
```

```
            IERR = N
            GO TO 99
         END IF
         AA(1) = SQRT(AA(1))

         CALL NDPUTR (NAMEA,DSNAMA,KNL,KNU,1,AA,MAXC,1,IERR)
         IF (IERR.NE.0) GO TO 93
80    CONTINUE
      RETURN

93    IERR = -3
99    CALL NMSERR ('NMVDD4')
      RETURN
      END


************************************************** SUBROUTINE NMVDF2 *****

      SUBROUTINE NMVDF2 (NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                   NE,MAXC,NR,AA,IA,RR,NCOL,IERR)

C
C--- Function : Internal routine controls data transfer and calls
C---            routine "NMVDF3" to perform forkward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, trans(U) X = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER*(*)  NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX
      DIMENSION      AA(1),IA(1),RR(1)

      IIST = 1
      NCOL1 = NCOL + 1
10    IIEND = IIST + NCOL
      IF (IIEND .GT. NE+1) THEN
         IIEND = NE + 1
         NCOL1 = IIEND - IIST + 1
      END IF
      CALL NDGETR (NAMEI,DSNAMI,IIST,IIEND,1,IA,NCOL1,1,IERR)
      IF (IERR.NE.0) GO TO 93

      IAST = IA(1)
      IAEND = IA(NCOL1) - 1
      IALEN = IAEND - IAST + 1
      CALL NDGETR (NAMEU,DSNAMU,IAST,IAEND,1,AA,IALEN,1,IERR)
      IF (IERR.NE.0) GO TO 93
```

```
      IRST = IIST - MAXC + 1
      IF (IRST .LT. 1) IRST = 1
      IREND = IIEND - 1
      IRLEN = IREND - IRST + 1
      IRIDX = IIST - IRST + 1
      NCOL = NCOL1 - 1

      DO 20 K = 1,NR
         ITR = K
         CALL NDGETR (NAMEX,DSNAMX,IRST,IREND,ITR,RR,IRLEN,1,IERR)
         IF (IERR.NE.0) GO TO 93
         CALL NMVDF3 (AA,IA,RR(IRIDX),NCOL,ITR)
         CALL NDPUTR (NAMEX,DSNAMX,IIST,IREND,ITR,RR(IRIDX),NCOL,1,
     *                IERR)
         IF (IERR.NE.0) GO TO 93
20    CONTINUE

      IF (IIEND .LT. NE+1) THEN
         IIST = IIEND
         GO TO 10
      ELSE
         RETURN
      END IF

93    IERR = -3
99    CALL NMSERR ('NMVDF2')
      RETURN
      END


********************************************** SUBROUTINE NMVDF3 *****

      SUBROUTINE NMVDF3 (AA,IA,RR,NCOL,ITR)

C
C--- Function : Internal routine to perform forward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, trans(U) X = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            This routine is called by "NMVDF2" which does data
C---            transfer.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION  AA(1),IA(1),RR(1)
      DATA       ZERO/0.D0/

C-- reset the index array IA.
```

```
      IF (ITR.EQ.1) THEN
         IDELTA = IA(1) - 1
         DO 10 I = 1,NCOL+1
            IA(I) = IA(I) - IDELTA
10       CONTINUE
      END IF

C-- forward substitution.

      DO 30 N = 1,NCOL
         KD = IA(N)
         KL = KD + 1
         KU = IA(N+1) - 1
         KR = N
         C  = ZERO

         DO 20 KK = KL,KU
            KR = KR - 1
            C = C + AA(KK) * RR(KR)
20       CONTINUE
         RR(N) = (RR(N) - C) / AA(KD)
30    CONTINUE

      RETURN
      END


********************************************** SUBROUTINE NMVDB2 *****

      SUBROUTINE NMVDB2 (NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                   NE,MAXC,NR,AA,IA,RR,NCOL,IERR)

C
C--- Function : Internal routine controls data transfer and calls
C---            routine "NMVDB3" to perform backward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, UX = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER*(*)  NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX
      DIMENSION      AA(1),IA(1),RR(1)

      IIEND = NE + 1
      NCOL1 = NCOL + 1
10    IIST = IIEND - NCOL
      IF (IIST .LT. 1) THEN
         IIST = 1
```

```
         NCOL1 = IIEND
      END IF
      CALL NDGETR (NAMEI,DSNAMI,IIST,IIEND,1,IA,NCOL1,1,IERR)
      IF (IERR.NE.0) GO TO 93

      IAST = IA(1)
      IAEND = IA(NCOL1) - 1
      IALEN = IAEND - IAST + 1
      CALL NDGETR (NAMEU,DSNAMU,IAST,IAEND,1,AA,IALEN,1,IERR)
      IF (IERR.NE.0) GO TO 93

      IRST = IIST - MAXC + 1
      IF (IRST .LT. 1) IRST = 1
      IREND = IIEND - 1
      IRLEN = IREND - IRST + 1
      IRIDX = IIST - IRST + 1
      NCOL = NCOL1 - 1

      DO 20 K = 1,NR
         ITR = K
         CALL NDGETR (NAMEX,DSNAMX,IRST,IREND,ITR,RR,IRLEN,1,IERR)
         IF (IERR.NE.0) GO TO 93
         CALL NMVDB3 (AA,IA,RR(IRIDX),NCOL,ITR)
         CALL NDPUTR (NAMEX,DSNAMX,IRST,IREND,ITR,RR,IRLEN,1,IERR)
         IF (IERR.NE.0) GO TO 93
20    CONTINUE

      IF (IIST .GT. 1) THEN
         IIEND = IIST
         GO TO 10
      ELSE
         RETURN
      END IF

93    IERR = -3
99    CALL NMSERR ('NMVDB2')
      RETURN
      END


********************************************** SUBROUTINE NMVDB3 *****

      SUBROUTINE NMVDB3 (AA,IA,RR,NCOL,ITR)

C
C--- Function : Internal routine to backform forward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, UX = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            This routine is called by "NMVDB2" which does data
```

```
C---            transfer.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION  AA(1),IA(1),RR(1)
      DATA       ZERO/0.D0/

C-- reset the index array IA.

      IF (ITR.EQ.1) THEN
         IDELTA = IA(1) - 1
         DO 10 I = 1,NCOL+1
            IA(I) = IA(I) - IDELTA
10       CONTINUE
      END IF

C-- backward substitution.

      DO 30 N = NCOL,1,-1
         KD = IA(N)
         RR(N) = RR(N) / AA(KD)
         KL = KD + 1
         KU = IA(N+1) - 1
         KR = N
         C  = ZERO

         DO 20 KK = KL,KU
            KR = KR - 1
            RR(KR) = RR(KR) - RR(N)*AA(KK)
20       CONTINUE
30    CONTINUE

      RETURN
      END
```

APPENDIX B

EQUATION SOLVER OF TYPE B

```
C-----------------------------------------------------------------------
C SMART N-325
C SUBROUTINE SKYSOLB (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
C                     NAMEX,DSNAMX,NE,MAXC,NR,WKSP,NWK,IERR)
C
C Function : Solves the symmetric variable band width system of
C            equations, AX = R.  The input matrix A is stored in
C            one-dimensional form as one-column data set DSNAMA.
C            Input index array I containing indices of diagonal
C            elements locations of matrix A is stored in data set
C            DSNAMI.  Input right hand side column vectors R(NE,NR)
C            are stored in data set DSNAMR.  The input data are
C            undisturbed on return.  The subroutine creates an output
C            data set DSNAMX for storing output column vectors
C            X(NE,NR). This subroutine uses only the minimum required
C            workin space.
C
C Arguments: NAMEA  - CHARACTER*(*) - I
C                     name of the database containing the data set
C                     DSNAMA.
C            DSNAMA - CHARACTER*(*) - I
C                     name of the data set containing input symmetric
C                     variable band width matrix A.  The data in the
C                     data set is factorized on return.  The data
C                     set must be a column vector.  But, it can be
C                     row wise or column wise.  Its dimension needs
C                     to be greater than or equal to NT x 1, NT is
C                     total elements number of matrix A.  Data type
C                     can be real or double precision real.
C            NAMEI  - CHARACTER*(*) - I
C                     name of the database to store the data set
C                     DSNAMI.
C            DSNAMI - CHARACTER*(*) - I
C                     name of the data set to store index array
C                     I(NE+1).  The data in the data set is
C                     undisturbed on return.  The data set must be a
C                     column vector with data type of four bytes
C                     integer.  But, it can be row wise or column
C                     wise.  Its dimension needs to be greater than
C                     or equal to NE+1 x 1.
C            NAMER  - CHARACTER*(*) - I
C                     name of the database containing the data set
C                     DSNAMR.
C            DSNAMR - CHARACTER*(*) - I
C                     name of the data set containing right hand side
C                     column vectors R(NE,NR).  The data set should
C                     have same data type as DSNAMU.  Suggested data
C                     set order is column wise.  The data in the data
C                     set is undisturbed on return.
C            NAMEX  - CHARACTER*(*) - I
C                     name of the database to create the data set
```

```
C                       DSNAMX.
C             DSNAMX - CHARACTER*(*) - I
C                       name of the data set to store the solution
C                       column vectors X(NE,NR).  X is solved by using
C                       Choleski solution algorithm.  Data set name is
C                       input.  The data set created is an NE x NR
C                       column data set with data type same as data set
C                       DSNAMR.
C             NE      - INTEGER*4 - I
C                       number of columns of matrix A in full matrix
C                       form.  Default value is that the number of rows
C                       of data set DSNAMI minus one.
C             MAXC    - INTEGER*4 - I
C                       the maximum band width of variable band width
C                       matrix A.  Default value is calculated from
C                       index array I.
C             WKSP    - REAL*4 or REAL*8
C                       working space for the subroutine.  Its data type
C                       needs to be same as data set DSNAMA.
C             NWK     - INTEGER*4 - I
C                       dimension of the working space.  Minimum
C                       requirement is 3MAXC for real data type, 2.5MAXC
C                       for double precision real data type.
C             IERR    - INTEGER*4 - O
C                       error code.  On return,
C                       IERR = 0 means successful solving;
C                       IERR > 0 means singular A matrix; value of IERR
C                       contains the position of singularity;
C                       IERR = -1 means working space is too small;
C                       IERR = -2 means wrong data set dimension or
C                       wrong data type;
C                       IERR = -3 means error message from MIDAS/N.
C
C Note :  i) The solution can be stored under the data set name
C             contained in DSNAMR.  This can be done when NAMEX and
C             DSNAMX are the same as NAMER and DSNAMR.  If this is the
C             case, then DSNAMR must be a column data set.
C-----------------------------------------------------------------------

      SUBROUTINE SKYSOLB (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
     *                    NAMEX,DSNAMX,NE,MAXC,NR,WKSP,NWK,IERR)

      CHARACTER*(*)  NAMEA,DSNAMA,NAMEI,DSNAMI,NAMER,DSNAMR,
     *               NAMEX,DSNAMX
      REAL           WKSP(1)

C-- Check input arguments and data sets

      NER = NE
      MAXCR = MAXC
      CALL NMVCKI (NAMEI,DSNAMI,NER,MAXCR,NT,WKSP,NWK,IERR)
```

```
      IF (IERR.NE.0) GO TO 99

      CALL NMVCKA (NAMEA,DSNAMA,NT,ICDTY,IERR)
      IF (IERR.NE.0) GO TO 99

      IF (ICDTY.EQ.4) THEN
         MINIWK = MAXCR * 3
      ELSE
         MINIWK = MAXCR * 5 / 2
      END IF
      IF (MINIWK.GT.NWK) THEN
         IERR = -1
         GO TO 99
      END IF

      NRR = NR
      CALL NMACRX (NAMEX,DSNAMX,NAMER,DSNAMR,NER,NRR,ICDTY,IERR)
      IF (IERR.NE.0) GO TO 99

C-- Call internal decomposition routine depend on data type

      IF (ICDTY.EQ.4) THEN
         N2 = MAXCR + 1
         N3 = N2 + MAXCR - 1
         CALL NMVSDP (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                WKSP,WKSP(N2),WKSP(N3),IERR)
         IF (IERR.NE.0) GO TO 99
         CALL NMVSFS (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                NER,MAXCR,NRR,WKSP,WKSP(N2),IERR)
         IF (IERR.NE.0) GO TO 99
         CALL NMVSBS (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                NER,MAXCR,NRR,WKSP,WKSP(N2),IERR)

      ELSE
         N2 = MAXCR*2 + 1
         N3 = N2 + (MAXCR-1)*2
         CALL NMVDDP (NAMEA,DSNAMA,NAMEI,DSNAMI,NER,MAXCR,
     *                WKSP,WKSP(N2),WKSP(N3),IERR)
         IF (IERR.NE.0) GO TO 99
         CALL NMVDFS (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                NER,MAXCR,NRR,WKSP,WKSP(N2),IERR)
         IF (IERR.NE.0) GO TO 99
         CALL NMVDBS (NAMEA,DSNAMA,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                NER,MAXCR,NRR,WKSP,WKSP(N2),IERR)
      END IF

      IF (IERR.EQ.0) RETURN

99    CALL NMIERR (IERR)
      CALL NMSERR ('SKYSOLB')
      RETURN
```

```
      END

********************************************** SUBROUTINE NMACRX ******

      SUBROUTINE NMACRX (NAMEX,DSNAMX,NAMER,DSNAMR,NE,NR,ICDTY,IERR)

C
C--- Function : Check input data set DSNAMR and creat output data
C---            set DSNAMX with required dimension for forward and
C---            backward substitution of system of equations.
C

      CHARACTER*(*)  NAMEX,DSNAMX,NAMER,DSNAMR,ORDER*4,DTYPE*4

C-- Check data type, storage order,and dimensions of data set DSNAMR

      CALL NDSARG (NAMER,DSNAMR,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IERR)
      IF (IERR.NE.0) GO TO 93

      IF (NE.GT.NROW) THEN
         GO TO 92
      ELSE IF (ORDER.NE.'ROW' .AND. ORDER.NE.'COL') THEN
         GO TO 92
      ELSE IF (DTYPE.EQ.'REAL') THEN
         IF (ICDTY.NE.4) GO TO 92
      ELSE IF (DTYPE.EQ.'DREA') THEN
         IF (ICDTY.NE.5) GO TO 92
      ELSE
         GO TO 92
      END IF

      IF (NR.EQ.0) THEN
         NR = NCOL
      ELSE IF (NR.GT.NCOL) THEN
         GO TO 92
      END IF

C-- Check whether or not the DSNAMX and DSNAMR are the same data set

      IF (NAMEX.EQ.NAMER .AND. DSNAMX.EQ.DSNAMR) THEN
         RETURN
      END IF

C-- if not, creat output data set DSNAMX

      CALL NDSDUP (NAMER,DSNAMR,NAMEX,DSNAMX,IERR)
      IF (IERR.NE.0) GO TO 93

      CALL NDSRDF (NAMEX,DSNAMX,0,0,'COL',NE,NR,' ',IERR)
      IF (IERR.EQ.0) RETURN
```

```
92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMACRX')
      RETURN
      END


********************************************** SUBROUTINE NMIERR *****

      SUBROUTINE NMIERR (IERR)

C
C--- Function : Report the error message of the error code.
C

      IF (IERR.GT.0) THEN
         WRITE (1,2000) IERR
2000     FORMAT (//
     *    ' ERROR - STIFFNESS MATRIX NOT POSITIVE DEFINITE;'/
     *    ' NONPOSTIVE PIVOT AT EQUATION ',I4/)
      ELSE IF (IERR.EQ.-1) THEN
         WRITE (1,2100)
2100     FORMAT (/
     *    ' ERROR -1 : WORKING SPACE IS TOO SMALL.'/)
      ELSE IF (IERR.EQ.-2) THEN
         WRITE (1,2200)
2200     FORMAT (/
     *    ' ERROR -2 : WRONG DATA SET DIMENSION OR WRONG DATA TYPE.'/)
      ELSE IF (IERR.EQ.-3) THEN
         WRITE (1,2300)
2300     FORMAT (/
     *    ' ERROR -3 : ERROR MESSAGE FROM MIDAS/N.'/)
      ELSE
         WRITE (1,2400) IERR
2400     FORMAT (/
     *    ' NO ERROR MESSAGE AVAILABLE UNDER THIS ERROR CODE :',I5/)
      END IF
      RETURN
      END


********************************************** SUBROUTINE NMSERR *****

      SUBROUTINE NMSERR (SUBNAM)

C
C--- Function : Write error subroutine name to terminal.
C
```

```
      CHARACTER  SUBNAM*(*)

      WRITE (1,2000) SUBNAM
2000  FORMAT ('--- the error comes from subroutine "',A,'".'/)
      RETURN
      END


********************************************** SUBROUTINE NMVCKA ******

      SUBROUTINE NMVCKA (NAMEA,DSNAMA,NT,ICDTY,IERR)

C
C--- Function : Check the coeficient matrix data set DSNAMA of the
C---            symmetric variable band width system of equations
C---            and get its data type code.
C

      CHARACTER  NAMEA*(*), DSNAMA*(*), ORDER*4, DTYPE*4

C-- Check data type, storage order, and dimensions of the data set

      IERR = 0
      CALL NDSARG (NAMEA,DSNAMA,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IERR)
      IF (IERR.NE.0) THEN
         GO TO 93
      ELSE IF (ORDER.NE.'ROW' .AND. ORDER.NE.'COL') THEN
         GO TO 92
      ELSE IF (NT .GT. NROW) THEN
         GO TO 92
      END IF

      IF (DTYPE.EQ.'REAL') THEN
         ICDTY = 4
      ELSE IF (DTYPE.EQ.'DREA') THEN
         ICDTY = 5
      ELSE
         GO TO 92
      END IF
      RETURN

92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMVCKA')
      RETURN
      END
```

```
********************************************** SUBROUTINE NMVCKI ******

      SUBROUTINE NMVCKI (NAMEI,DSNAMI,NE,MAXC,NT,IWKSP,NWK,IERR)

C
C--- Function : Check the index array data set DSNAMI and get
C---            total elements number of the coefficient matric.
C

      CHARACTER  NAMEI*(*), DSNAMI*(*), ORDER*4, DTYPE*4
      INTEGER*4  IWKSP(1)

C-- Check data type, storage order, and dimensions of the data set

      IERR = 0
      CALL NDSARG (NAMEI,DSNAMI,ISUB,JSUB,ORDER,NROW,NCOL,DTYPE,IER2)
      IF (IER2.NE.0) THEN
         GO TO 93
      ELSE IF (DTYPE.NE.'INTL') THEN
         GO TO 92
      ELSE IF (ORDER.NE.'ROW'.AND.ORDER.NE.'COL') THEN
         GO TO 92
      ELSE IF (NE.EQ.0) THEN
         NE = NROW - 1
      ELSE IF (NE.GE.NROW.OR.NE.LT.0) THEN
         GO TO 92
      END IF

C-- Get the maximum column height and
C-- the total number of elements of the coefficient matrix

      NE1 = NE + 1
      IF (MAXC.EQ.0) THEN
         ILEN = NWK
         IST = 1
5        IEND = IST + ILEN - 1
         IF (IEND.GT.NE1) THEN
            IEND = NE1
            ILEN = IEND - IST + 1
         END IF
         CALL NDGETR (NAMEI,DSNAMI,IST,IEND,1,IWKSP,ILEN,1,IER2)
         IF (IER2.NE.0) GO TO 93
         DO 10 I = ILEN,2,-1
            IDEF = IWKSP(I) - IWKSP(I-1)
            IF (IDEF .GT. MAXC) MAXC = IDEF
10       CONTINUE
         IF (IEND.LT.NE1) THEN
            IST = IEND
            GO TO 5
         END IF
         NT = IWKSP(ILEN) - 1
```

```
      ELSE
         CALL NDGETR (NAMEI,DSNAMI,NE1,NE1,1,NT,1,1,IER2)
         IF (IER2.NE.0) GO TO 93
         NT = NT - 1
      END IF

C-- Check the maximum column height and required working space

      IF (MAXC*NE .LT. NT) THEN
         GO TO 92
      ELSE IF (NWK .LT. MAXC*2) THEN
         IERR = -1
         GO TO 99
      END IF
      RETURN

92    IERR = -2
      GO TO 99

93    IERR = -3
99    CALL NMSERR ('NMVCKI')
      RETURN
      END


************************************************** SUBROUTINE NMVDBS ******

      SUBROUTINE NMVDBS (NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX,
     *                   NE,MAXC,NR,AA,RR,IERR)

C
C--- Function : Internal routine to perform backward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, UX = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            (for double precision real data type)
C

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER*(*)  NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX
      DIMENSION      AA(1),RR(1)

      DO 80 IR = 1,NR
         NE1 = NE + 1
         CALL NDGETR (NAMEI,DSNAMI,NE1,NE1,1,IA2,1,1,IERR)
         IF (IERR.NE.0) GO TO 93
         IA2 = IA2 - 1

         ISELM = NE - MAXC + 1
         CALL NDGETC (NAMEX,DSNAMX,IR,IR,ISELM,RR,MAXC,1,IERR)
```

```
         IF (IERR.NE.0) GO TO 93

         DO 50 N = NE,1,-1
            CALL NDGETR (NAMEI,DSNAMI,N,N,1,IA1,1,1,IERR)
            IF (IERR.NE.0) GO TO 93

            KH = IA2 - IA1 + 1
            CALL NDGETR (NAMEU,DSNAMU,IA1,IA2,1,AA,MAXC,1,IERR)
            IF (IERR.NE.0) GO TO 93

            RRC = RR(MAXC) / AA(1)
            RR(MAXC) = RRC
            K = MAXC
            DO 20 I = 2,KH
               K = K - 1
               RR(K) = RR(K) - RRC*AA(I)
20          CONTINUE
            CALL NDPUTC (NAMEX,DSNAMX,IR,IR,N,RR(MAXC),1,1,IERR)
            IF (IERR.NE.0) GO TO 93

            DO 30 I = MAXC,2,-1
               RR(I) = RR(I-1)
30          CONTINUE
            ISELM = ISELM - 1
            IF (ISELM.GT.0) THEN
               CALL NDGETC (NAMEX,DSNAMX,IR,IR,ISELM,RR,1,1,IERR)
               IF (IERR.NE.0) GO TO 93
            END IF
            IA2 = IA1 - 1
50       CONTINUE
80    CONTINUE
      RETURN

93    IERR = -3
99    CALL NMSERR ('NMVDBS')
      RETURN
      END


********************************************** SUBROUTINE NMVDDP *****

      SUBROUTINE NMVDDP (NAMEA,DSNAMA,NAMEI,DSNAMI,NE,MAXC,
     *                   AA,BB,IA,IERR)

C
C--- Function : Internal routine to decompose a symmetric variable
C---            band width matrix, A = trans(U) * U.  The input
C---            data sets must be checked before calling this
C---            routine.  The decomposed coeficient matrix is stored
C---            in input coefficient matrix on return.
C---            (for double precision real data type)
```

```
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER*(*) NAMEA,DSNAMA,NAMEI,DSNAMI
      DIMENSION     AA(1),BB(1),IA(1)
      DATA          ZERO/0.DO/,AMIN/1.D-30/

      MAXI  = MAXC - 1
      MAXC1 = MAXC + 1
      CALL NDGETR (NAMEI,DSNAMI,1,1,1,IA(MAXC1),1,1,IERR)
      IF (IERR.NE.0) GO TO 93

      DO 80 N = 1,NE
         N1 = N + 1
         DO 10 I = 1,MAXC
            IA(I) = IA(I+1)
10       CONTINUE

         CALL NDGETR (NAMEI,DSNAMI,N1,N1,1,IA(MAXC1),1,1,IERR)
         IF (IERR.NE.0) GO TO 93

         KNL = IA(MAXC)
         KNU = IA(MAXC1) - 1
         KNH = KNU - KNL
         CALL NDGETR (NAMEA,DSNAMA,KNL,KNU,1,AA,MAXC,1,IERR)
         IF (IERR.NE.0) GO TO 93

         KNP = KNH + 1
         KIAC = MAXC - KNH
         IC = 0
         DO 50 I = 1,KNH
            KIL = IA(KIAC)
            KIH = IA(KIAC+1) - KIL - 1
            KK  = MIN (IC,KIH)
            KIU = KIL + KK
            CALL NDGETR (NAMEA,DSNAMA,KIL,KIU,1,BB,MAXI,1,IERR)
            IF (IERR.NE.0) GO TO 93

            C = ZERO
            DO 20 J = 1,KK
               C = C + AA(KNP+J)*BB(1+J)
20          CONTINUE

            AA(KNP) = (AA(KNP)-C) / BB(1)
            IC = IC + 1
            KNP = KNP - 1
            KIAC = KIAC + 1
50       CONTINUE

         C = ZERO
         DO 60 J = 1,KNH
```

```
            C = C + AA(1+J)**2
60        CONTINUE

          AA(1) = AA(1) - C
          IF (AA(1).LT.AMIN) THEN
             IERR = N
             GO TO 99
          END IF
          AA(1) = SQRT(AA(1))

          CALL NDPUTR (NAMEA,DSNAMA,KNL,KNU,1,AA,MAXC,1,IERR)
          IF (IERR.NE.0) GO TO 93
80      CONTINUE
        RETURN

93      IERR = -3
99      CALL NMSERR ('NMVDDP')
        RETURN
        END


********************************************** SUBROUTINE NMVDFS *****

        SUBROUTINE NMVDFS (NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX,
       *                   NE,MAXC,NR,AA,RR,IERR)

C
C--- Function : Internal routine to perform forward substitution
C---            to solve the decomposed symmetric variable band
C---            width system of equations, trans(U) X = R.
C---            The solution vectors X are stored in input right
C---            hand side vectors on return.
C---            (for double precision real data type)
C

        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        CHARACTER*(*)  NAMEU,DSNAMU,NAMEI,DSNAMI,NAMEX,DSNAMX
        DIMENSION      AA(1),RR(1)
        DATA           ZERO/0.D0/

        DO 80 IR = 1,NR
           IA2 = 0

           DO 50 N = 1,NE
              N1 = N + 1
              IA1 = IA2 + 1
              CALL NDGETR (NAMEI,DSNAMI,N1,N1,1,IA2,1,1,IERR)
              IF (IERR.NE.0) GO TO 93

              DO 10 I = 2,MAXC
                 RR(I-1) = RR(I)
```

```
10          CONTINUE
            CALL NDGETC (NAMEX,DSNAMX,IR,IR,N,RR(MAXC),1,1,IERR)
            IF (IERR.NE.0) GO TO 93

            KH = IA2 - IA1
            IA2 = IA2 - 1
            CALL NDGETR (NAMEU,DSNAMU,IA1,IA2,1,AA,MAXC,1,IERR)
            IF (IERR.NE.0) GO TO 93

            C = ZERO
            K = MAXC
            DO 20 I = 2,KH
               K = K - 1
               C = C + AA(I) * RR(K)
20          CONTINUE
            RR(MAXC) = (RR(MAXC)-C) / AA(1)
            CALL NDPUTC (NAMEX,DSNAMX,IR,IR,N,RR(MAXC),1,1,IERR)
            IF (IERR.NE.0) GO TO 93
50       CONTINUE
80    CONTINUE
      RETURN

93    IERR = -3
99    CALL NMSERR ('NMVDFS')
      RETURN
      END
```

APPENDIX C

EQUATION SOLVER OF TYPE C

```
C-----------------------------------------------------------------------
C   SMART Z-225D
C   SUBROUTINE SKYSOLC (A,R,IAD,NE,NR,IER)
C   Function : Solves symmetric variable band system of equations,
C              AX = R, by decomposing A = trans(U) * D * U.
C   Arguments: A    - REAL*8
C                     half of symmetric variable band matrix stored
C                     in one-dimensional form at entry; decomposed
C                     upper triangular matrix and diagonal matrix on
C                     return.
C              R    - REAL*8
C                     right hand side array of size NE x NR at entry;
C                     solution array on return.
C              IAD  - INTEGER*4
C                     a vector containing addresses of diagonal
C                     elements of matrix A.
C              NE   - INTEGER*4
C                     order of matrix A.
C              NR   - INTEGER*4
C                     number of right hand side columns.
C              IER  - INTEGER*4
C                     IER = 0 on return means successful decomposition
C                     IER > 0 on return means singular A matrix, value
C                     of IER contains the position of singularity.
C-----------------------------------------------------------------------

      SUBROUTINE SKYSOLC (A,R,IAD,NE,NR,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(1),R(NE,1),IAD(1)

      CALL ZDVBDU (A,IAD,NE,IER)
      IF (IER.NE.0) RETURN
      CALL ZDVBFS (A,R,IAD,NE,NR)
      CALL ZDVBBS (A,R,IAD,NE,NR)

      RETURN
      END


C-----------------------------------------------------------------------
C   SMART Z-221D
C   SUBROUTINE ZDVBDU(A,IAD,NE,IER)
C   Function : Decomposes symmetric variable band matrix, A = trans(U)
C              * U. Decomposed matrix is stored in the original space;
C              original matrix is destoryed.
C   Arguments: A    - REAL*8
C                     half of symmetric variable band matrix stored
C                     in one-dimensional form at entry; decomposed
C                     upper triangular matrix on return.
C              IAD  - INTEGER*4
C                     a vector containing addresses of diagonal
```

```
C                     elements of matrix A.
C               NE    - INTEGER*4
C                     order of matrix A.
C               IER   - INTEGER*4
C                     IER = 0 on return means successful decomposition
C                     IER > 0 on return means singular A matrix, value
C                     of IER contains the position of singularity.
C-----------------------------------------------------------------------
      SUBROUTINE ZDVBDU(A,IAD,NE,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(1),IAD(1)

C     A(KL) : diagonal element in the Nth column.
C     A(KU) : highest element in the Nth column.
C     KH    : height of the Nth column.
C     KR    : row no. of current element in the Nth column.
C     KK    : no. of multiplications required in calculating A(KR,N).
C     Nth column stands for current column.
C
C     loop 1000 generates U, column by column, from left to right.
C
      IER=0
      DO 1000,N=1,NE
         KL=IAD(N)
         KU=IAD(N+1)-1
         KH=KU-KL
         IF(A(KL).LE.1.0D-15) THEN
           IER=N
           RETURN
         END IF
         KR=N-KH
         KLT=KU
         IC=0
C
C        loop 100 generates off-diagonal elements in the Nth column.
C
         DO 100,J=1,KH
            KI=IAD(KR)
            ND=IAD(KR+1)-KI-1
            KK=MIN0(IC,ND)
            C=0.D0
            DO 10,L=1,KK
               C=C+A(KI+L)*A(KLT+L)
10          CONTINUE
            A(KLT)=A(KLT)-C
            A(KLT)=A(KLT)/A(KI)
            IC=IC+1
            KR=KR+1
            KLT=KLT-1
100      CONTINUE
         C=0.D0
```

```
C
C        generates diagonal element in Nth column.
C
         DO 200,JJ=1,KH
            C=C+A(KL+JJ)*A(KL+JJ)
200      CONTINUE
         A(KL)=A(KL)-C
         IF(A(KL).LE.1.0D-15) THEN
           IER=N
           RETURN
         END IF
         A(KL)=DSQRT(A(KL))
1000  CONTINUE

      RETURN
      END


C-----------------------------------------------------------------------
C   SMART Z-222D
C   SUBROUTINE ZDVBFS(A,R,IAD,NE,NR)
C   Function : Performs forward substitution to solve decomposed
C              symmetric variable band system of equations, AX = R,
C              A is generated by Z-221D.
C   Arguments: A     - REAL*8
C                      half of decomposed symmetric variable band
C                      matrix stored in one-dimensional form.
C              R     - REAL*8
C                      right hand side array of size NE x NR at entry;
C                      solution array on return.
C              IAD   - INTEGER*4
C                      a vector containing addresses of diagonal
C                      elements of matrix A.
C              NE    - INTEGER*4
C                      order of matrix A.
C              NR    - INTEGER*4
C                      number of right hand side columns.
C-----------------------------------------------------------------------

      SUBROUTINE ZDVBFS(A,R,IAD,NE,NR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(1),IAD(1),R(NE,1)

C
C     A(KD) : diagonal element in the Nth column.
C     A(KL) : element just above the diagonal in the Nth column.
C     A(KU) : highest element in the Nth column.
C
C     loop 2000 generates R(1),R(2),.....,R(NE).
C
      DO 1000,NS=1,NR
```

```
         DO 2000,N=1,NE
            KD=IAD(N)
            KL=KD+1
            KU=IAD(N+1)-1
            K=N
            C=0.D0
            DO 100,KK=KL,KU
               K=K-1
               C=C+A(KK)*R(K,NS)
100         CONTINUE
            R(N,NS)=R(N,NS)-C
            R(N,NS)=R(N,NS)/A(KD)
2000     CONTINUE
1000  CONTINUE

      RETURN
      END


C-----------------------------------------------------------------------
C   SMART Z-223D
C   SUBROUTINE ZDVBBS(A,R,IAD,NE,NR)
C   Function : Performs backward substitution to solve decomposed
C              symmetric variable band system of equations, AX = R,
C              A is generated by Z-221D.
C   Arguments: A    - REAL*8
C                     half of decomposed symmetric variable band
C                     matrix stored in one-dimensional form.
C              R    - REAL*8
C                     right hand side array of size NE x NR at entry;
C                     solution array on return.
C              IAD  - INTEGER*4
C                     a vector containing addresses of diagonal
C                     elements of matrix A.
C              NE   - INTEGER*4
C                     order of matrix A.
C              NR   - INTEGER*4
C                     number of right hand side columns.
C-----------------------------------------------------------------------

      SUBROUTINE ZDVBBS(A,R,IAD,NE,NR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(1),IAD(1),R(NE,1)

C
C     A(KD) : diagonal element in Nth column.
C     A(KL) : the element just above diagonal in Nth column.
C     A(KU) : the highest element in Nth column.
C     Nth column stands for current column.
C
C     loop 2000 generates R(NE),R(NE-1),.....,R(1).
```

```
C
      DO 1000,NS=1,NR
         DO 2000,N=NE,1,-1
            KD=IAD(N)
            R(N,NS)=R(N,NS)/A(KD)
            IF(N.EQ.1) RETURN
            KL=KD+1
            KU=IAD(N+1)-1
            K=N
            C=0.D0
            DO 100,KK=KL,KU
               K=K-1
               R(K,NS)=R(K,NS)-R(N,NS)*A(KK)
100         CONTINUE
2000     CONTINUE
1000  CONTINUE

      RETURN
      END
```

APPENDIX D

PERFORMANCE DATA OF EQUATION SOLVER SKYSOLA

CA ; 1 : System Parameters : NPAGES = 20, IPSIZE = 256 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	8.773	70	30	4054
25	500	10	17.737	160	62	8127
25	1000	10	35.515	340	127	16270
50	1000	10	21.530	340	127	7690
100	1000	10	17.621	340	127	5203
200	1000	10	7.707	339	127	843
400	1000	10	6.268	339	127	356
600	1000	10	6.139	340	127	225
800	1000	10	5.798	340	127	167
1000	1000	10	5.700	340	128	134
2000	1000	10	5.465	368	138	67
3000	1000	10	5.403	370	138	45
4000	1000	10	5.419	364	136	34
5000	1000	10	5.336	357	135	34
6000	1000	10	5.365	360	133	23
7000	1000	10	5.228	351	132	20
8000	1000	10	5.207	342	130	12
9000	1000	10	5.245	342	130	12
10000	1000	10	5.250	342	130	12
3000	1000	30	28.618	1499	519	134
3000	1000	50	56.481	3007	646	433
3000	5000	50	293.280	15516	3279	2194

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	19.967	71	30	4054
25	500	10	41.061	161	62	8127
25	1000	10	82.973	341	127	16270
50	1000	10	42.356	341	127	7690
100	1000	10	31.247	341	127	5203
200	1000	10	10.056	341	127	843
400	1000	10	7.378	342	127	356
600	1000	10	6.726	344	127	225
800	1000	10	6.348	341	127	167
1000	1000	10	6.250	356	128	134
2000	1000	10	6.000	403	138	67
3000	1000	10	6.060	480	138	45
4000	1000	10	6.057	484	136	34
5000	1000	10	5.964	472	135	34
6000	1000	10	5.892	471	133	23
7000	1000	10	5.855	464	132	20
8000	1000	10	5.692	456	130	12
9000	1000	10	5.709	456	130	12
10000	1000	10	5.714	456	130	12
3000	1000	30	31.802	2006	519	134

CASE 2 : System Parameters : NPAGES = 80, IPSIZE = 256 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	8.145	34	29	4054
25	500	10	16.628	64	57	8127
25	1000	10	35.758	275	119	16270
50	1000	10	21.732	275	119	7690
100	1000	10	17.500	275	119	5203
200	1000	10	7.426	275	119	843
400	1000	10	6.028	275	119	356
600	1000	10	5.593	275	119	225
800	1000	10	5.465	276	119	167
1000	1000	10	5.259	276	120	134
2000	1000	10	5.214	277	121	67
3000	1000	10	5.108	280	121	45
4000	1000	10	5.217	270	124	34
5000	1000	10	5.150	256	122	34
6000	1000	10	5.077	246	121	23
7000	1000	10	5.162	242	120	20
8000	1000	10	4.884	226	130	12
9000	1000	10	4.817	226	130	12
10000	1000	10	4.797	226	130	12
3000	1000	30	29.348	1334	478	134
3000	1000	50	55.577	1694	598	433
3000	5000	50	278.340	8897	3039	2194

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	26.547	34	29	4054
25	500	10	66.126	64	57	8127
25	1000	10	164.256	276	119	16270
50	1000	10	78.825	276	119	7690
100	1000	10	55.257	276	119	5203
200	1000	10	14.645	277	119	843
400	1000	10	9.483	277	119	356
600	1000	10	8.141	278	119	225
800	1000	10	7.534	279	119	167
1000	1000	10	7.181	280	120	134
2000	1000	10	6.520	291	121	67
3000	1000	10	6.247	289	121	45
4000	1000	10	6.301	310	124	34
5000	1000	10	6.166	297	122	34
6000	1000	10	6.011	289	121	23
7000	1000	10	6.414	400	120	20
8000	1000	10	6.531	455	130	12
9000	1000	10	6.534	455	130	12
10000	1000	10	6.553	455	130	12
3000	1000	30	34.060	1373	478	134

CASE 3 : System Parameters : NPAGES = 20, IPSIZE = 1024 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	8.018	12	8	4054
25	500	10	16.118	19	15	8127
25	1000	10	33.285	70	30	16270
50	1000	10	19.121	70	30	7690
100	1000	10	15.143	70	30	5203
200	1000	10	6.303	70	30	843
400	1000	10	5.078	70	30	356
600	1000	10	4.796	70	30	225
800	1000	10	4.774	70	30	167
1000	1000	10	4.709	70	30	134
2000	1000	10	4.499	71	30	67
3000	1000	10	4.455	73	30	45
4000	1000	10	4.471	70	31	34
5000	1000	10	4.335	65	31	34
6000	1000	10	4.382	65	30	23
7000	1000	10	4.417	69	31	20
8000	1000	10	4.327	61	33	12
9000	1000	10	4.261	61	33	12
10000	1000	10	4.268	61	33	12
3000	1000	30	24.657	337	120	134
3000	1000	50	48.532	427	150	433
3000	5000	50	249.305	2229	761	2194

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	15.555	12	8	4054
25	500	10	34.189	19	15	8127
25	1000	10	75.191	71	30	16270
50	1000	10	38.513	71	30	7690
100	1000	10	27.930	71	30	5203
200	1000	10	8.444	71	30	843
400	1000	10	6.021	71	30	356
600	1000	10	5.377	71	30	225
800	1000	10	5.089	72	30	167
1000	1000	10	4.939	72	30	134
2000	1000	10	4.620	73	30	67
3000	1000	10	4.506	74	30	45
4000	1000	10	4.599	79	31	34
5000	1000	10	4.456	76	31	34
6000	1000	10	4.443	75	30	23
7000	1000	10	4.627	109	33	20
8000	1000	10	4.622	121	33	12
9000	1000	10	4.531	121	33	12
10000	1000	10	4.527	121	33	12
3000	1000	30	25.478	353	120	134

CASE 4 : System Parameters : NPAGES = 80, IPSIZE = 1024 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	8.242	12	8	4054
25	500	10	16.808	19	15	8127
25	1000	10	33.628	34	29	16270
50	1000	10	19.737	34	29	7690
100	1000	10	15.491	34	29	5203
200	1000	10	6.287	34	29	843
400	1000	10	4.909	34	29	356
600	1000	10	4.628	34	29	225
800	1000	10	4.430	34	29	167
1000	1000	10	4.322	34	29	134
2000	1000	10	4.137	34	29	67
3000	1000	10	4.076	34	29	45
4000	1000	10	4.318	34	29	34
5000	1000	10	4.061	34	29	34
6000	1000	10	4.145	34	29	23
7000	1000	10	4.082	34	29	20
8000	1000	10	4.026	34	29	12
9000	1000	10	4.023	34	29	12
10000	1000	10	4.025	34	29	12
3000	1000	30	24.759	275	118	134
3000	1000	50	49.078	365	148	433
3000	5000	50	252.376	2168	759	2194

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	21.733	12	8	4054
25	500	10	46.559	19	15	8127
25	1000	10	106.513	34	29	16270
50	1000	10	53.016	34	29	7690
100	1000	10	37.162	34	29	5203
200	1000	10	10.040	34	29	843
400	1000	10	6.631	34	29	356
600	1000	10	5.882	34	29	225
800	1000	10	5.427	34	29	167
1000	1000	10	5.152	34	29	134
2000	1000	10	4.571	34	29	67
3000	1000	10	4.377	34	29	45
4000	1000	10	4.414	34	29	34
5000	1000	10	4.344	34	29	34
6000	1000	10	4.236	34	29	23
7000	1000	10	4.247	34	29	20
8000	1000	10	4.173	34	29	12
9000	1000	10	4.176	34	29	12
10000	1000	10	4.169	34	29	12
3000	1000	30	26.693	285	118	134

APPENDIX E

PERFORMANCE DATA OF EQUATION SOLVER SKYSOLB

CASE 1 : System Parameters : NPAGES = 20, IPSIZE = 256 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	7.921	70	30	4046
25	500	10	16.279	160	62	8119
25	1000	10	33.056	340	127	16262
75	1000	30	125.369	1394	480	38688
125	1000	50	220.169	18366	1164	46334
125	3000	50	664.040	57123	3583	140432

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	18.289	71	30	4046
25	500	10	37.912	161	62	8119
25	1000	10	75.733	341	127	16262
75	1000	30	232.577	1395	480	38688

CASE 2 : System Parameters : NPAGES = 80, IPSIZE = 256 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	7.543	34	29	4046
25	500	10	15.720	64	57	8119
25	1000	10	34.074	275	119	16262
75	1000	30	126.700	1333	478	38688
125	1000	50	180.879	1693	598	46334
125	3000	50	555.117	5290	1817	140432

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	25.413	34	29	4046
25	500	10	63.710	64	57	8119
25	1000	10	155.828	276	119	16262
75	1000	30	478.048	1334	478	38688

CASE 3 : System Parameters : NPAGES = 20, IPSIZE = 1024 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	7.579	12	8	4046
25	500	10	15.325	19	15	8119
25	1000	10	30.690	70	30	16262
75	1000	30	114.737	337	120	38688
125	1000	50	162.255	427	150	46334
125	3000	50	489.802	1326	455	140432

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	14.498	12	8	4046
25	500	10	31.790	19	15	8119
25	1000	10	69.011	71	30	16262
75	1000	30	220.350	338	120	38688

CASE 4 : System Parameters : NPAGES = 80, IPSIZE = 1024 Words

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (2.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	7.729	12	8	4046
25	500	10	15.702	19	15	8119
25	1000	10	31.881	34	29	16262
75	1000	30	123.491	274	118	38688
125	1000	50	167.689	365	148	46334
125	3000	50	514.473	1265	453	140432

Working Space Units: REAL*8	Eq. Size NEQ	MAXC	-----MIDAS/N (1.0)----- CPU-time	Nread	Nwrite	No. of Calls to MIDAS/N
25	250	10	20.521	12	8	4046
25	500	10	43.534	19	15	8119
25	1000	10	101.797	34	29	16262
75	1000	30	416.577	275	118	38688

APPENDIX F

ONE WAY TO USE THE INTERACTIVE SUBROUTINE NDQURY

Step 1. Prepare a short program to call subroutine NDQURY.

```
OK, ED
INPUT
C************************************** INTERACTIVE PROGRAM ******
C
      CALL NDQURY
C
      STOP
      END

EDIT
FILE TEST
OK,
```

Step 2. Compile and load the short program with SMART library.

```
OK, F77 TEST -DEBUG
[F77 Rev. 19.2.6]
0000 ERRORS [<.MAIN.> F77-REV 19.2.6]
OK, SEG
[SEG rev 19.2.2]
# VLOAD #TEST
$ LOAD B_TEST
$ LIB SMART.LIB
$ LIB VAPPLB
$ LIB
LOAD COMPLETE
$ MAP 6

$ SAV
$ QUIT
OK,
```

Step 3. Run the program interactively.

```
OK, SEG #TEST
----- available commands are as follows :
      NDBCMP    NDBDEL    NDBDFN    NDBEND    NDBINF
      NDBOPN    NDBRNM    NDGETC    NDGETM    NDGETR
      NDINFO    NDJASN    NDPUTC    NDPUTM    NDPUTR
      NDSARG    NDSCPY    NDSDEL    NDSDFN    NDSGET
      NDSINF    NDSPUT    NDSRDF    NDSRNM    AUTO-M
      MANU-M    QUIT      SHOWDS

>>>>> Enter command ?(A6)
(hit return for command listing)
NDBDFN
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the pathname ?
    (CHARACTER*40  PTHNAME)
SHYY>MIDAS>TEST
Enter the type of the database file ?
    (CHARACTER* 4  TYPE)
RA
Enter the status of the database ?
    (CHARACTER* 4  STAT)
PERM

****** given arguments of command NDBDFN are as follows ******
 1: database name ------------------------- NAME = DATABASE_A
 2: pathname ---------------------------- PTHNAME = SHYY>MIDAS>TEST
 3: type of the database file ------------- TYPE = RA
 4: status of the database ----------------- STAT = PERM

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

****** Given command NDBDFN has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDSDFN
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET
Enter the dimensions of the submatrix ?
    ( INTEGER  ISUB,JSUB )
```

```
0, 0
Enter the data set order ?
    (CHARACTER* 4  ORDER)
ROW
Enter the dimensions of the data set ?
    ( INTEGER  NROW,NCOL )
8, 9
Enter the data type ?
    (CHARACTER* 4  DTYPE)
INTL

***** given arguments of command NDSDFN are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: data set name ------------------------ DSNAME = DATA_SET
 3: dimensions of the submatrix ------- ISUB,JSUB =      0      0
 4: data set order ------------------------ ORDER = ROW
 5: dimensions of the data set -------- NROW,NCOL =      8      9
 6: data type ----------------------------- DTYPE = INTL

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDSDFN has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDSPUT
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET
Enter the starting row or column no. ?
    ( INTEGER  NSTR )
1
Enter the ending row or column no. ?
    ( INTEGER  NEND )
8
Enter the starting element number ?
    ( INTEGER  ISTR )
1
Enter the data set order ?
    (CHARACTER* 4  ORDER)
ROW
Enter the dimensions of user's buffer ?
    ( INTEGER  IROW,ICOL )
8, 9
```

```
***** given arguments of command NDSPUT are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: data set name ------------------------ DSNAME = DATA_SET
 3: starting row or column no. ------------- NSTR =      1
 4: ending row or column no. --------------- NEND =      8
 5: starting element number ---------------- ISTR =      1
 6: data set order ------------------------ ORDER = ROW
 7: dimensions of user's buffer ------- IROW,ICOL =      8     9

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

          COL 1  COL 2  COL 3  COL 4  COL 5  COL 6  COL 7  COL 8  COL 9

ROW 1 :      11     12     13     14     15     16     17     18     19
ROW 2 :      21     22     23     24     25     26     27     28     29
ROW 3 :      31     32     33     34     35     36     37     38     39
ROW 4 :      41     42     43     44     45     46     47     48     49
ROW 5 :      51     52     53     54     55     56     57     58     59
ROW 6 :      61     62     63     64     65     66     67     68     69
ROW 7 :      71     72     73     74     75     76     77     78     79
ROW 8 :      81     82     83     84     85     86     87     88     89

***** Given command NDSPUT has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
SHOWDS
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET

***** given arguments of command SHOWDS are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: data set name ------------------------ DSNAME = DATA_SET

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

          COL 1  COL 2  COL 3  COL 4  COL 5  COL 6  COL 7  COL 8  COL 9

ROW 1 :      11     12     13     14     15     16     17     18     19
ROW 2 :      21     22     23     24     25     26     27     28     29
ROW 3 :      31     32     33     34     35     36     37     38     39
ROW 4 :      41     42     43     44     45     46     47     48     49
```

```
ROW 5 :      51     52     53     54     55     56     57     58     59
ROW 6 :      61     62     63     64     65     66     67     68     69
ROW 7 :      71     72     73     74     75     76     77     78     79
ROW 8 :      81     82     83     84     85     86     87     88     89

***** Given command SHOWDS has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDGETR
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET
Enter the starting row or column no. ?
    ( INTEGER  NSTR )
3
Enter the ending row or column no. ?
    ( INTEGER  NEND )
6
Enter the starting element number ?
    ( INTEGER  ISTR )
3
Enter the dimensions of user's buffer ?
    ( INTEGER  IROW,ICOL )
6, 8

***** given arguments of command NDGETR are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: data set name ------------------------- DSNAME = DATA_SET
 3: starting row or column no. ------------ NSTR =      3
 4: ending row or column no. -------------- NEND =      6
 5: starting element number --------------- ISTR =      3
 6: dimensions of user's buffer ------- IROW,ICOL =      6     8

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

          COL 1  COL 2  COL 3  COL 4  COL 5  COL 6  COL 7  COL 8  COL 9

ROW 1 :      33     34     35     36     37     38     39      0
ROW 2 :      43     44     45     46     47     48     49      0
ROW 3 :      53     54     55     56     57     58     59      0
ROW 4 :      63     64     65     66     67     68     69      0
ROW 5 :       0      0      0      0      0      0      0      0
ROW 6 :       0      0      0      0      0      0      0      0
```

```
***** Given command NDGETR has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDBDFN
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B
Enter the pathname ?
    (CHARACTER*40  PTHNAME)

Enter the type of the database file ?
    (CHARACTER* 4  TYPE)
SA
Enter the status of the database ?
    (CHARACTER* 4  STAT)
TEMP

***** given arguments of command NDBDFN are as follows *****
 1: database name ------------------------- NAME = DATABASE_B
 2: pathname ------------------------------ PTHNAME =
 3: type of the database file -------------- TYPE = SA
 4: status of the database ----------------- STAT = TEMP

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDBDFN has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDINFO


*** Currently opened database.
 1 : DATABASE_A
 2 : DATABASE_B
*** Currently  2 databases are opened.



***** Given command NDINFO has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDSCPY
Enter the database name ?
```

```
    (CHARACTER*20  NAME)
DATABASE_A
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B

***** given arguments of command NDSCPY are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: data set name ------------------------ DSNAME = DATA_SET
 3: database name ------------------------- NAME = DATABASE_B

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDSCPY has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDBINF
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B

***** given arguments of command NDBINF are as follows *****
 1: database name ------------------------- NAME = DATABASE_B

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0


*** Currently defined data sets in database : DATABASE_B
 1 : DATA_SET
*** Currently  1 data set are defined.



***** Given command NDBINF has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDSRDF
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B
```

```
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET
Enter the dimensions of the submatrix ?
    ( INTEGER  ISUB,JSUB )
3, 2
Enter the data set order ?
    (CHARACTER* 4  ORDER)
SUB
Enter the dimensions of the data set ?
    ( INTEGER  NROW,NCOL )
9, 6
Enter the data type ?
    (CHARACTER* 4  DTYPE)
DREA

***** given arguments of command NDSRDF are as follows *****
 1: database name ------------------------- NAME = DATABASE_B
 2: data set name ------------------------ DSNAME = DATA_SET
 3: dimensions of the submatrix ------- ISUB,JSUB =      3     2
 4: data set order ------------------------ ORDER = SUB
 5: dimensions of the data set -------- NROW,NCOL =      9     6
 6: data type ----------------------------- DTYPE = DREA

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDSRDF has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
SHOWDS
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET

***** given arguments of command SHOWDS are as follows *****
 1: database name ------------------------- NAME = DATABASE_B
 2: data set name ------------------------ DSNAME = DATA_SET

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

          COL 1  COL 2  COL 3  COL 4  COL 5  COL 6  COL 7  COL 8  COL 9
```

```
ROW 1 :     11.     12.     13.     14.     15.     16.
ROW 2 :     21.     22.     23.     24.     25.     26.
ROW 3 :     31.     32.     33.     34.     35.     36.
ROW 4 :     41.     42.     43.     44.     45.     46.
ROW 5 :     51.     52.     53.     54.     55.     56.
ROW 6 :     61.     62.     63.     64.     65.     66.
ROW 7 :     71.     72.     73.     74.     75.     76.
ROW 8 :     81.     82.     83.     84.     85.     86.
ROW 9 :      0.      0.      0.      0.      0.      0.

***** Given command SHOWDS has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDSINF
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B
Enter the data set name ?
    (CHARACTER*12  DSNAME)
DATA_SET

***** given arguments of command NDSINF are as follows *****
 1: database name ------------------------- NAME = DATABASE_B
 2: data set name ------------------------- DSNAME = DATA_SET

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0
     THE DATA SET NAME IS : DATA_SET
     STARTING LOCATION IS :        628
     NUMBER OF WORDS   IS :        216
     SUBMATRIX DIMENSION  :     3,    2
     DATA SET DIMENSION   :     9,    6
     DATA SET ORDERING IS : SUB
     THE DATA TYPE IS     : DREA

***** Given command NDSINF has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDBEND
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A

***** given arguments of command NDBEND are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
```

```
enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDBEND has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDBEND
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B

***** given arguments of command NDBEND are as follows *****
 1: database name ------------------------- NAME = DATABASE_B

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDBEND has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
NDBOPN
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A
Enter the pathname ?
    (CHARACTER*40  PTHNAME)
SHYY>MIDAS>TEST
Enter the database access right (R/W/U) ?
    (CHARACTER* 4  ACCESS)
U

***** given arguments of command NDBOPN are as follows *****
 1: database name ------------------------- NAME = DATABASE_A
 2: pathname ------------------------------ PTHNAME = SHYY>MIDAS>TEST
 3: database access right (R/W/U) -------- ACCESS = U

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDBOPN has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
```

```
NDBOPN
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_B
Enter the pathname ?
    (CHARACTER*40  PTHNAME)

Enter the database access right (R/W/U) ?
    (CHARACTER* 4  ACCESS)
U

***** given arguments of command NDBOPN are as follows *****
 1: database name ------------------------- NAME = DATABASE_B
 2: pathname ---------------------------- PTHNAME =
 3: database access right (R/W/U) -------- ACCESS = U

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0
--- the error comes from subroutine "NIFOPN".

 ERROR 207 - the given database does not exist

--- the error comes from subroutine "NDBOPN".


????? error during perform the command NDBOPN


>>>>> Enter command ?(A6)
(hit return for command listing)
NDINFO


*** Currently opened database.
 1 : DATABASE_A
*** Currently  1 databases are opened.



????? error during perform the command NDINFO


>>>>> Enter command ?(A6)
(hit return for command listing)

----- available commands are as follows :
      NDBCMP    NDBDEL    NDBDFN    NDBEND    NDBINF
      NDBOPN    NDBRNM    NDGETC    NDGETM    NDGETR
      NDINFO    NDJASN    NDPUTC    NDPUTM    NDPUTR
      NDSARG    NDSCPY    NDSDEL    NDSDFN    NDSGET
```

```
      NDSINF     NDSPUT     NDSRDF     NDSRNM     AUTO-M
      MANU-M     QUIT       SHOWDS

>>>>> Enter command ?(A6)
(hit return for command listing)
NDBEND
Enter the database name ?
    (CHARACTER*20  NAME)
DATABASE_A

***** given arguments of command NDBEND are as follows *****
 1: database name ------------------------- NAME = DATABASE_A

enter the leading number to modify that argument,
      0 - to perform the job, 99 - to cancel the job.
0

***** Given command NDBEND has been done.


>>>>> Enter command ?(A6)
(hit return for command listing)
QUIT
**** STOP
OK,
```

END

1-87

DTIC